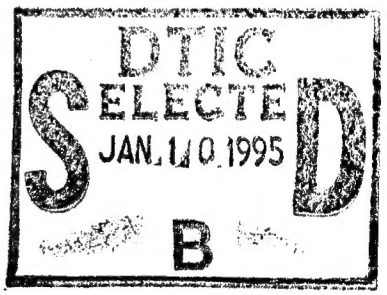
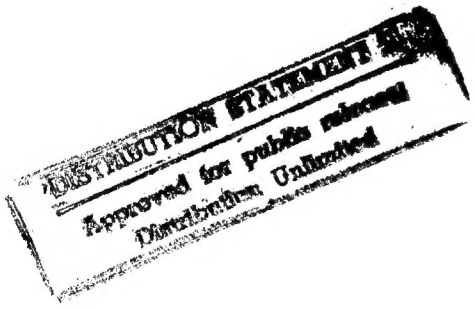
TASK: PV03
CDRL: A023
03 December 1993

# Process Instrumentation Process (PIP)and Amadeus Guidelines Version 0.5 - Draft

Informal Technical Data




STARS-VC-A023/001/00
03 December 1993

# REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188



| 1. AGENCY USE ONLY (Leave blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | 03 December 1993 | Informal Technical Report |

**4. TITLE AND SUBTITLE**

Process Instrumentation
Process (PIP) and Amadeus Guidelines
Version 0.5 - Draft

**5. FUNDING NUMBERS**

F19628-93-C-0130

**6. AUTHOR(S)**

Aaron Goldstein - TRW

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

**8. PERFORMING ORGANIZATION REPORT NUMBER**

STARS-VC-A023/001/00

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Department of the Air Force
Headquarters, Electronics System Center

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

A023

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Distribution "A"

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

This document is an initial draft description of the STARS Process Instrumentation Process (PIP) and guidelines for applying the Amadeus Measurement System to that process. It is a product of Paramax STARS U-Increment efforts to explore automated support for software process measurement. These efforts have focused primarily on the Amadeus Measurement System.

**14. SUBJECT TERMS**

**15. NUMBER OF PAGES**
55

**16. PRICE CODE**

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | SAR |

TASK: PV03
CDRL: A023
3 December 1993

INFORMAL TECHNICAL REPORT

For

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

*Process Instrumentation Process (PIP)*
*and Amadeus Guidelines Version 0.5*
*(Draft)*

STARS-VC-A023/001/00
3 December 1993

Data Type: Informal Technical Data

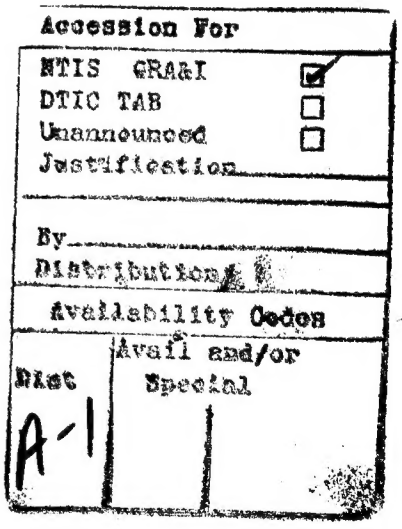
CONTRACT NO. F19628-93-C-0130

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

TRW
under contract to
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091



TASK: PV03
CDRL: A023
3 December 1993

Data ID: STARS-VC-A023/001/00

Distribution Statement "A"
per DoD Directive 5230.24
Authorized for public release; Distribution is unlimited.



Developed by: TRW under contract to
Unisys Corporation

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Scheme (DoD Directive 5230.24) unless otherwise indicated. Sponsored by the U.S. Advanced Research Projects Agency (ARPA) under contract F19628-88-D-0031, the STARS program is supported by the military services, SEI, and MITRE, with the U.S. Air Force as the executive contracting agent.

TASK: PV03
CDRL: A023
3 December 1993

INFORMAL TECHNICAL REPORT
Process Instrumentation Process (PIP)
and Amadeus Guidelines Version 0.5
(Draft)

**Principal Author(s):**

---

*Aaron Goldstein*

**Approvals:**

---

Chief Programmer *Hal Hart* *Date*

Teri Payton 12/6/93

Program Manager *Teri Payton* *Date*

*(Signatures on File)*

## Contents

**1 Introduction** **1**
1.1 Identification . . . . . 1
1.2 Scope . . . . . 1
1.3 Organization . . . . . 1

**2 Process Model for The PIP** **3**
2.1 Context . . . . . 3
2.2 Node A0: Instrument Software Process . . . . . 8
2.3 Node A1: Determine Measurement Requirements . . . . . 11
2.4 Node A2: Design Measurement Mechanisms . . . . . 14
2.4.1 Node A21: Design Automated Mechanisms . . . . . 17
2.4.2 Node A22: Design Manual Mechanisms . . . . . 20
2.4.3 Node A23: Evaluate Cost of Proposed Designs . . . . . 22
2.5 Node A3: Implement Measurement Mechanisms . . . . . 24
2.5.1 Node A31: Acquire Measurement Tools . . . . . 26
2.6 Node A4: Execute Measurement Mechanisms . . . . . 28

**3 Amadeus Guidelines** **30**
3.1 Determining Measurement Requirements . . . . . 30
3.2 Designing Measurement Mechanisms . . . . . 30
3.2.1 Designing Automated Mechanisms . . . . . 31
3.2.2 Designing Manual Mechanisms . . . . . 32
3.2.3 Evaluating Cost of Proposed Designs . . . . . 32
3.3 Implementing Measurement Mechanisms . . . . . 33
3.3.1 Acquiring Measurement Tools . . . . . 33
3.3.2 Integrating Measurement Tools . . . . . 34
3.4 Executing Measurement Mechanisms . . . . . 35

**A Amadeus Overview** **37**
A.1 Data Collection . . . . . 39
A.1.1 Monitoring The Environment . . . . . 40
A.1.2 Triggering Agents . . . . . 40
A.1.3 Entering Data Interactively . . . . . 40
A.1.4 Importing Data from Foreign Tools . . . . . 41
A.2 Metrics Visualization and Reporting . . . . . 41
A.2.1 Extracting Data from Amadeus' Database . . . . . 41
A.2.2 Formatting The Data . . . . . 41
A.2.3 Generating Graphs . . . . . 42
A.3 Exporting Data to Foreign Tools . . . . . 42
A.4 Metrics Analysis/Integration . . . . . 42
A.4.1 Classification Analysis . . . . . 42
A.5 Systematic Feedback and Empirical Guidance . . . . . 43

**B Glossary** **44**

B.1 IDEF0 Terminology . . . . . 44
B.2 PIP Process Model Terminology . . . . . 45
B.3 Amadeus Terminology . . . . . 53

**C PIP Decomposition** **55**

**List of Figures**


1 Context Diagram . . . . . 4
2 Decomposition of Instrument Software Process . . . . . 8
3 Decomposition of Determine Measurement Requirements . . . . . 11
4 Decomposition of Design Measurement Mechanisms . . . . . 14
5 Decomposition of Design Automated Mechanisms . . . . . 17
6 Decomposition of Design Manual Mechanisms . . . . . 20
7 Decomposition of Evaluate Cost of Proposed Designs . . . . . 22
8 Decomposition of Implement Measurement Mechanisms . . . . . 24
9 Decomposition of Acquire Measurement Tools . . . . . 26
10 Decomposition of Execute Measurement Mechanisms . . . . . 28
11 Amadeus Control and Data Flow . . . . . 38
12 PIP Decomposition Hierarchy . . . . . 55

# 1 Introduction

## 1.1 Identification

This document is an initial draft description of the STARS Process Instrumentation Process (PIP) and guidelines for applying the Amadeus[1] Measurement System to that process. It is a product of Paramax STARS U-Increment efforts to explore automated support for software process measurement. These efforts have focused primarily on the Amadeus Measurement System.

## 1.2 Scope

The purpose of this document is to guide future users of software measurement technology in the application of that technology. It describes a generic process for instrumenting a software engineering process, and the software engineering environment (SEE) that supports it, to automate software measurement (i.e., collection, analysis, reporting and feedback of software measurement data – also known as software metrics). This process is referred to as the Process Instrumentation Process (PIP). It encompasses not only the planning and implementation of software measurement mechanisms, but sustained execution of those mechanisms as well. The PIP is generic in the sense that it does not depend on any particular measurement technology. In fact, it allows for implementation of manual software measurement mechanisms as well as automated mechanisms – although it favors the use of automated mechanisms whenever possible and cost-effective. As an example of how to apply a particular software measurement technology in support of the PIP, this document provides guidelines for applying the Amadeus Measurement System (for brevity, we will refer to this as simply Amadeus from now on).

This document does not provide detailed instructions on how to use Amadeus. Such instructions can be found in the user documentation that is provided by Amadeus Software Research as part of the Amadeus commercial product.

## 1.3 Organization

The body of this document consists of two main sections. Section 2 presents a diagrammatic process model representing the PIP. Section 3 presents a set of guidelines for applying Amadeus to the PIP. Both of these sections are organized hierarchically, according to the hierarchical decomposition of the PIP.

The document also includes a set of appendices containing information intended for reference. Appendix A contains an overview of Amadeus capabilities. Appendix B contains definitions of technical terms used in this document (Note: if a term is in italics or underlined, its definition can be found in this appendix). Appendix C contains an illustration of the PIP

[1] Amadeus is a trademark of Amadeus Software Research

decomposition hierarchy.

## 2 Process Model for The PIP

The process model for the PIP is represented using a diagrammatic notation known as IDEF0. IDEF stands for Integrated Computer Aided Manufacturing (ICAM) Definition Method. IDEF0 is one of several IDEF modeling paradigms, each suited to modeling a different kind of system and each using a different technique for system representation. IDEF0 is a static modeling paradigm that represents a system as a network of interconnected activities. This representation technique makes IDEF0 particularly useful for modeling software lifecycle processes.

An IDEF0 model represents a process as a hierarchy of *activities*. It has a single activity at the top of the hierarchy that represents the overall process. This activity is decomposed into subactivities and each of the subactivities may be further decomposed into subsubactivities and so on. Each decomposition of an activity into its immediate subactivities is depicted on a separate page of the model, called a *decomposition page*. Every model also has a page depicting just the single top-level activity, called a *context page*.

Each activity has a set of associated *inputs*, *controls*, *outputs* and *mechanisms*, referred to in general as *ICOMs*. The activity on the context page has ICOMs representing the context in which the process is assumed to operate (i.e., the inputs, controls and mechanisms available to the process as a whole and the outputs that it in turn makes available to other processes). The activities on each decomposition page are linked together by connecting their ICOMs (e.g., by connecting the output of one activity to an input, control or mechanism of another).

Each page of an IDEF0 model contains an IDEF0 diagram. In these diagrams the activities are depicted as boxes and the ICOMs are depicted as arrows connecting the activity boxes. Each activity box has an associated name that identifies the activity and a *node number* that identifies its place in the activity hierarchy. Similarly, each ICOM has an identifying label. The side of an activity box from which an arrow enters or leaves determines whether it is an input, control, output or mechanism: inputs always enter an activity box from the left side; controls always enter an activity box from the top; outputs always leave an activity box from the right and mechanisms always enter an activity box from the bottom. The point where an ICOM attaches to an activity box is referred to as a *port*. If the activity is decomposed, the ports are represented by corresponding ICOM labels around the edges of the decomposition page; these labels are tagged with a port number, so that they can be easily matched with the ports on the activity box for the *parent activity*. To reduce clutter, a feature called a *tunnel* may be applied to a port, making the corresponding ICOM invisible on the other side of the port. For further descriptions of IDEF0 terminology and notation, refer to the glossary in Appendix B (specifically, Section B.1).

### 2.1 Context

Figure 1 shows the context page for the PIP. IDEF0 conventions require that all activity names be verb phrases, so the activity on the context page is named *Instrument Software Process*, rather than Process Instrumentation Process. As explained above, this single ac-

tivity represents the overall process that is being modeled.

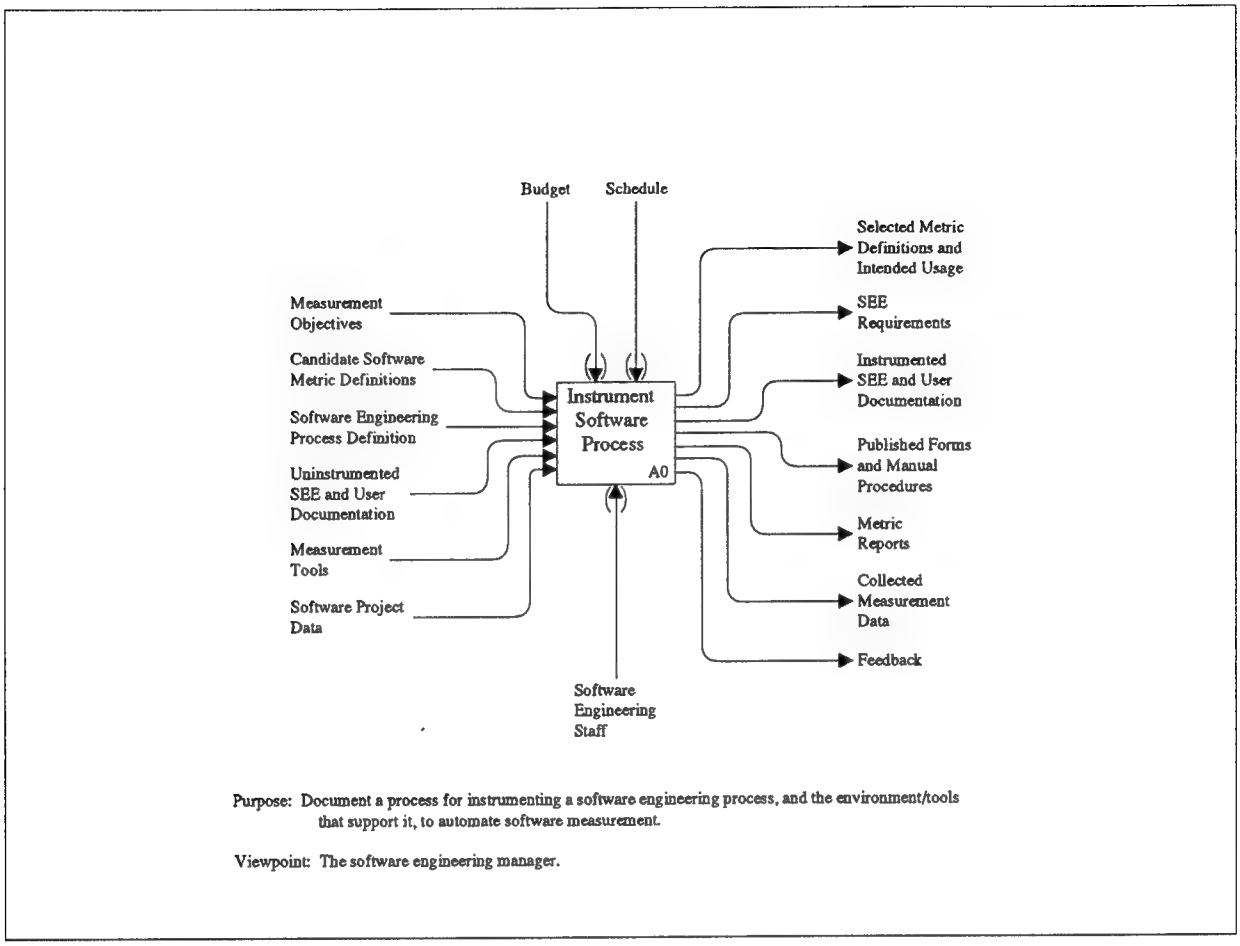



Figure 1: Context Diagram

The Instrument Software Process activity is by no means a standalone activity. It has interfaces with many other software engineering activities, including software process definition, SEE construction, software development and maintenance, software project management and software process improvement. These interfaces are manifested as ICOMs on the context page.

The *budget* and *schedule* controls represent budget and schedule constraints placed on the overall process of instrumenting a software process. These normally come from software project management activities.

The *measurement objectives* input represents specific statements of what objectives are to be achieved by software measurement. These may come from software project management activities, or they may come from software process improvement activities. They are typically derived from a combination of customer requirements and/or expectations, corporate goals, and project goals or needs.

The *candidate software metric definitions* input represents existing definitions of software

metrics that may be used to achieve the measurement objectives. For some metrics, there may be several different definitions from which to choose. In such cases, customers or corporate organizations may dictate which definitions are acceptable candidates, based on their own biases. These candidate metric definitions usually come from either a customer's software acquisition activities or a corporate organization's software process improvement activities.

The *software engineering process definition* input represents descriptions of the software engineering lifecycle processes to which measurement will be applied. These need not be formal process descriptions. However, greater accuracy and detail generally enable more effective application of software measurement. This input may come from software process definition activities; or alternatively, it may come from software development and maintenance activities, such as writing a software development plan.

The *uninstrumented SEE and user documentation* input represents the software engineering environment (SEE) and its accompanying documentation before the SEE has been instrumented with automated mechanisms to support software measurement. If the Instrument Software Process activity is performed iteratively, this input may not represent an entirely uninstrumented SEE; so the term, "uninstrumented," may be misleading. It is actually being used in a relative sense, rather than an absolute sense. We can say that the uninstrumented SEE is relatively uninstrumented, when compared with the instrumented SEE that is produced by the Instrument Software Process activity. The uninstrumented SEE and its accompanying user documentation usually come from SEE construction activities (also known as SEE integration activities).

The *measurement tools* input represents existing automated tools that may be used to collect, analyze, report and/or feedback measurement data. These need not be tools that are specifically designed for measurement. Certain kinds of general-purpose tools can often be adapted to perform measurement functions as well. For example, a spreadsheet with a reporting capability could be adapted to generate metric reports. Measurement tools may obtained from a variety of sources, including commercial tool vendors (for so-called COTS tools), government-sponsored research activities (for public domain tools) and/or in-house tool developers (for so-called proprietary tools). This may involve interfaces with corporate purchasing activities and/or customer software acquisition activities.

The *software project data* input represents all of the information generated by a software project. This includes software code, documentation, test results, problem reports, project organization charts, schedules, budgets, etc. – even collected measurement data. This information typically comes from software development and maintenance activities and software project management activities.

The *software engineering staff* mechanism represents all project personnel engaged in software engineering activities. This includes software designers, coders, testers, software engineering managers, software process engineers, etc. These personnel are normally obtained through corporate human resource management activities (e.g., hiring of staff or transfer from other projects).

The *sclected metric definitions and intended usage* output represents definitions of software metrics that have been chosen for collection and descriptions of how the metrics are to be used (i.e., analyzed, reported and fed back into the process being measured). This output may be incorporated into the software development plan as part of software development and maintenance activities.

The *SEE requirements* output represents descriptions of capabilities that the SEE must provide to support automation of software measurement. For example, if metrics concerning software problems are to be collected automatically, the SEE must provide a capability to track software problem reports on-line. The SEE requirements output should be provided as an input to SEE construction activities.

The *instrumented SEE and user documentation* output represents the SEE and its accompanying documentation after the SEE has been instrumented with automated mechanisms to support software measurement. The documentation includes instructions on how to use the automated measurement mechanisms that have been integrated into the SEE. The instrumented SEE and its accompanying user documentation should be provided as a mechanism to support software development and maintenance activities.

The *published forms and manual procedures* output represents data collection forms that have been reproduced in quantity (to support manual collection of measurement data) and published handbooks describing manual procedures for collection, analysis, reporting, interpretation and feedback of measurement data. Depending on the specific measurement objectives and the degree to which the SEE supports automation of software measurement, this output may or may not be produced; in some cases, no manual measurement mechanisms are required. This output should also be provided as a mechanism to support software development and maintenance activities.

The *metric reports* and *collected measurement data* outputs represent human-readable reports of software metrics and machine-readable raw measurement data, respectively. The metric reports may be textual or graphical and may include analysis results as well as raw measurement data. The collected measurement data may be in any format, but the format must be published so that the data can be read by external tools. These outputs should be supplied as inputs to software project management and software process improvement activities.

The *feedback* output represents information, derived from measurement data, which is used to control the process that is being measured. Feedback may be either manual (e.g., corrective action on the part of a software engineering manager) or automatic (e.g., execution of some automated software engineering tools). This output should be supplied as a control to whatever software engineering activities are appropriate, as dictated by the measurement objectives.

Some of the ICOMs on the context page (specifically, budget, schedule and software engineering staff) are tunneled, indicating that they are not central to the description of the PIP. Although these ICOMs still apply to all of the subactivities of the Instrument Software

Process activity, they are not shown explicitly on decomposition pages.

As we discuss each successive level of decomposition of the Instrument Software Process activity, in subsequent sections of this document, we will introduce a number of subactivities and additional ICOMs. These will be described only briefly where they are encountered. Note that a more detailed description of each activity and each ICOM is provided in the glossary in Appendix B (specifically, in Section B.2).

## 2.2 Node A0: Instrument Software Process

Figure 2 presents the first level of decomposition of the Instrument Software Process activity. According to this diagram, the Instrument Software Process consists of four subactivities: *Determine Measurement Requirements*, *Design Measurement Mechanisms*, *Implement Measurement Mechanisms* and *Execute Measurement Mechanisms*. These are analogous to the requirements, design, implementation and operation phases of the software lifecycle.

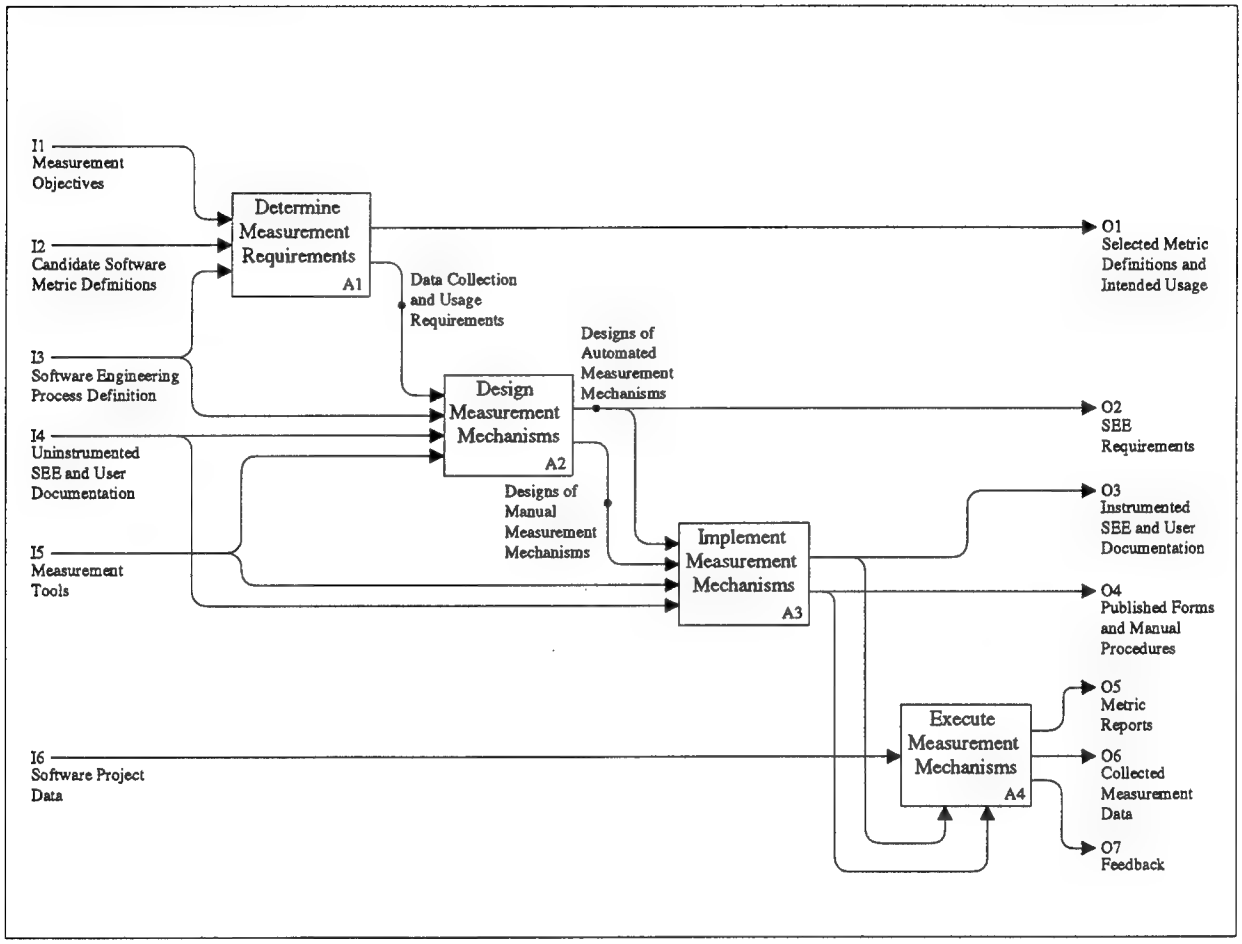



Figure 2: Decomposition of Instrument Software Process

The Determine Measurement Requirements activity is concerned with determining what to measure, when to measure it, and how and when to use it, in order to achieve a set of measurement objectives. It outputs this information as a set of *data collection and usage requirements*, which are subsequently input by the Design Measurement Mechanisms activity.

The Design Measurement Mechanisms activity is concerned with determining what measurement mechanisms (both automated and manual) should be used to meet the data collection and usage requirements. It produces two outputs: *designs of automated measurement mechanisms* and *designs of manual measurement mechanisms*. The former represents approved designs of automated measurement mechanisms (for collection, analysis, reporting and feedback of measurement data), which consist of a tool-independent design of each automated

mechanism, a description of any SEE capabilities required to support the mechanisms, a list of any existing measurement tools selected to implement the mechanisms, designs of any tools that need to be developed, and designs of any tool integration "glue" that needs to be developed. The latter represents approved designs of manual measurement mechanisms, which consist of designs of data collection forms for any manual data collection mechanisms and definitions of manual procedures for any manual data collection, analysis, reporting and feedback mechanisms. These designs of measurement mechanisms are supplied as inputs to the Implement Measurement Mechanisms activity.

The Implement Measurement Mechanisms activity is concerned with implementing the mechanisms according to the designs. The automated mechanisms are implemented by integrating appropriate measurement tools with the SEE (and updating the SEE documentation accordingly). The manual mechanisms are implemented by publishing the data collection forms and manual procedures. These implemented measurement mechanisms are then supplied to the Execute Measurement Mechanisms activity, which is essentially an integral part of the software development and maintenance activities.

Lastly, the Execute Measurement Mechanisms activity uses the implemented measurement mechanisms to achieve the measurement objectives.

To some extent, these activities must occur sequentially, since each depends on outputs produced by its predecessor. However, this need not be a strictly sequential process; each activity in the sequence need not complete before the next one starts. The measurement objectives may be achieved incrementally or they may evolve through successive iterations of the process. In fact, experience has shown that a measurement program that starts with a small set of metrics and expands gradually is much more likely to succeed than one that starts with a large set of metrics. This is mainly due to the substantial amount of planning that is required to determine how to collect, analyze and report each metric and feed the results back into the process being measured – not to mention the effort required to select the metrics in the first place. The key to success is to start with a modest set of measurement objectives and iterate the process to accommodate additional objectives later on.

At this first level of decomposition, we can start to see some of the details of how the Instrument Software Process activity fits together with other software engineering activities. The Determine Measurement Requirements and Design Measurement Mechanisms activities need to consider the software engineering processes where measurements are to be collected and used, so their inputs include a software engineering process defintion. This is presumably obtained from a separate software process definition activity. Similarly, the Design Measurement Mechanisms activity and Implement Measurement Mechanisms activity require the uninstrumented SEE and its corresponding user documentation as an input: the former activity for purposes of identifying physical sources and destinations of measurement data (i.e., users, tools and software artifacts from which data will be gathered or to which data will be supplied) and prototyping designs of measurement tools and tool integrations; the latter activity for purposes of implementing measurement tools and integrating such tools with the SEE. The uninstrumented SEE and user documentation are presumably pro-

duced by a separate SEE construction activity. Interestingly, this SEE construction activity must also take into account SEE requirements derived from the Design Measurement Mechanisms activity – specifically, requirements for capabilities to support measurement. Thus, some iteration (outside the scope of this model) is required between the SEE construction activity and the Design Measurement Mechanisms activity. The Implement Measurement Mechanisms activity takes the resulting uninstrumented SEE and user documentation, with the required support for measurement, as input. It produces an Instrumented SEE and corresponding user documentation, as well as published forms and manual procedures for measurement, that are presumably supplied as mechanisms (rather than inputs; note that they enter from the bottom of the activity box, rather than from the left side) for the software development and maintenance activities. The Execute Measurement Mechanisms activity is essentially an integral part of these software development and maintenance activities. It produces collected measurement data and metric reports that are presumably supplied as input to software project management and software process improvement activities. It also produces feedback that is presumably applied directly to the software engineering activities being measured.

Because the subactivities of the Instrument Software Process activity are so closely intertwined with other software engineering activities, they might even be folded into those other activities. For example, the Determine Measurement Requirements activity might be incorporated into software project management or software process improvement activities, the Design Measurement Mechanisms and Implement Measurement Mechanisms activities might be combined with SEE construction activities, and the Execute Measurement Mechanisms activity might integrated into software development and maintenance activities. There need not be a separate Instrument Software Process activity at all.

Nevertheless, we can still choose to model this collection of measurement activities as subactivities of a separate Instrument Software Process activity. This allows us to focus on activities related to software measurement without getting too caught up in the details of other software engineering activities. It is important to remember that this IDEF0 process model is merely an abstraction that we find useful for describing the PIP, and that the actual implementation of the PIP need not be structured in exactly the same way.

## 2.3 Node A1: Determine Measurement Requirements

Figure 3 presents the decomposition of the Determine Measurement Requirements activity. This activity consists of three subactivities: *Select Metrics*, *Identify Required Data Items* and *Identify Data Collection & Usage Points.*

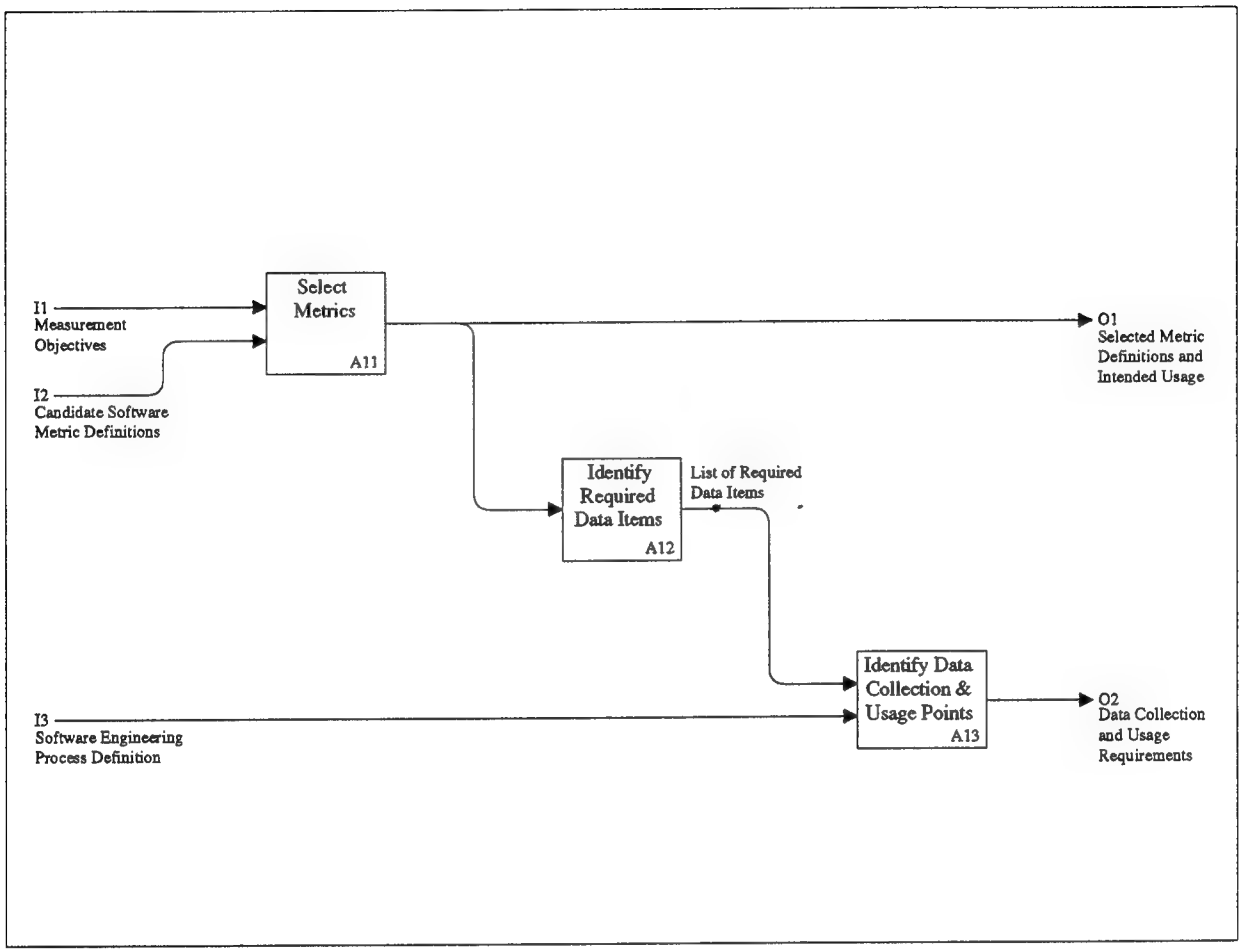



Figure 3: Decomposition of Determine Measurement Requirements

The Select Metrics activity is concerned with choosing appropriate software metrics to meet the measurement objectives. One approach that is often recommended for selecting software metrics is to apply Basili and Weiss' Goal-Question-Metric (GQM) paradigm. In this paradigm, the measurement objectives are refined in a stepwise fashion until they are detailed enough that specific questions can be framed about quantifiable characteristics of the software; these quantifiable characteristics constitute the selected metrics. It is important to consider not only what metrics to collect, but also the manner in which they will be used. Collection and storage of measurement data can be costly, so there is no point in collecting the data unless you plan to make use of it. A benefit of the GQM paradigm is that the process of refining the measurement objectives provides insight into how to use the collected metric data to achieve those objectives. Unfortunately, the GQM paradigm is strictly top-down, so it is difficult to apply when a set of candidate metric definitions is dictated. In that case,

it is usually best to work bottom-up, matching the documented purpose of each candidate metric against the measurement objectives. The output of the Select Metrics activity is a set of selected metric definitions and their intended usage. This is supplied as input to the Identify Required Data Items activity. It is also an overall output of the Instrument Software Process activity.

The Identify Required Data Items activity is concerned with identifying the individual pieces of measurement data that must be collected to enable calculation of the selected software metrics and to support use of those metrics as intended. When a measurement is made, it may be necessary to collect multiple pieces of information. This is clearly the case when a metric is defined as a formula involving multiple variables (e.g., defect density, which is a ratio of defects to lines of code). Another case where it may be necessary to collect multiple pieces of information is when the measurement data (or metric) is to be aggregated in various ways for analysis and/or reporting. For example, a source-lines-or-code (SLOC) metric may be aggregated by software component (at various levels of the component hierarchy) and also by programming language; so additional information identifying the software component and the programming language must be associated with the SLOC value when the measurement is performed. Sample metric reports (either textual or graphical) can be useful in identifying data items that are required to support aggregation. Sometimes, they can even be helpful in clarifying the definition of the metric (e.g., in cases where the textual definition of the metric is ambiguous). The output of the Identify Required Data Items activity is a *list of required data items.* This is supplied as an input to the Identify Data Collection & Usage Points activity.

The Identify Data Collection & Usage Points activity is concerned with identifying the points in the software engineering process where measurement data is to be collected and used, and also the frequency of collection and use. In this activity, the software engineering process definition is analyzed to determine the appropriate points (and frequency) for collecting the required data items, analyzing them, reporting the metrics and analysis results, and feeding them back into the process. This process definition need not be very formal. However, a clear and fairly detailed process definition generally reduces the amount of effort required for this activity. In considering where the measurement data is to be used, one may sometimes encounter confidentiality issues. Certain kinds of measurement data may be considered sensitive. In that case, access to the data must be controlled. Any such confidentiality issues must be documented at this point, so they can be addressed properly in the Design Measurement Mechanisms activity. The output of the Identify Data Collection & Usage Points activity consists of a list of data collection and usage requirements. It should include the following information for each metric:

- Metric Name
- Metric Definition (a very brief definition of the metric)
- Benefits (benefits accruing from the intended use)
- Algorithm/Formula (precise algorithm/formula for calculating the metric)

- Data Collection Requirements (what is to be collected when and from where)
- Data Usage Requirements (what data is to be used when and in what way and by whom)
- Sample Report Format (if usage includes reporting)

This output is supplied as the primary input for the Design Measurement Mechanisms activity.

Again, the dependencies between the activities impose a sequential ordering. But like the overall Instrument Software Process activity, this activity may also be executed iteratively, expanding the measurement objectives on each iteration.

## 2.4 Node A2: Design Measurement Mechanisms

Figure 4 presents the decomposition of the Design Measurement Mechanisms activity. This activity consists of three subactivities: *Design Automated Mechanisms*, *Design Manual Mechanisms* and *Evaluate Cost of Proposed Designs.*

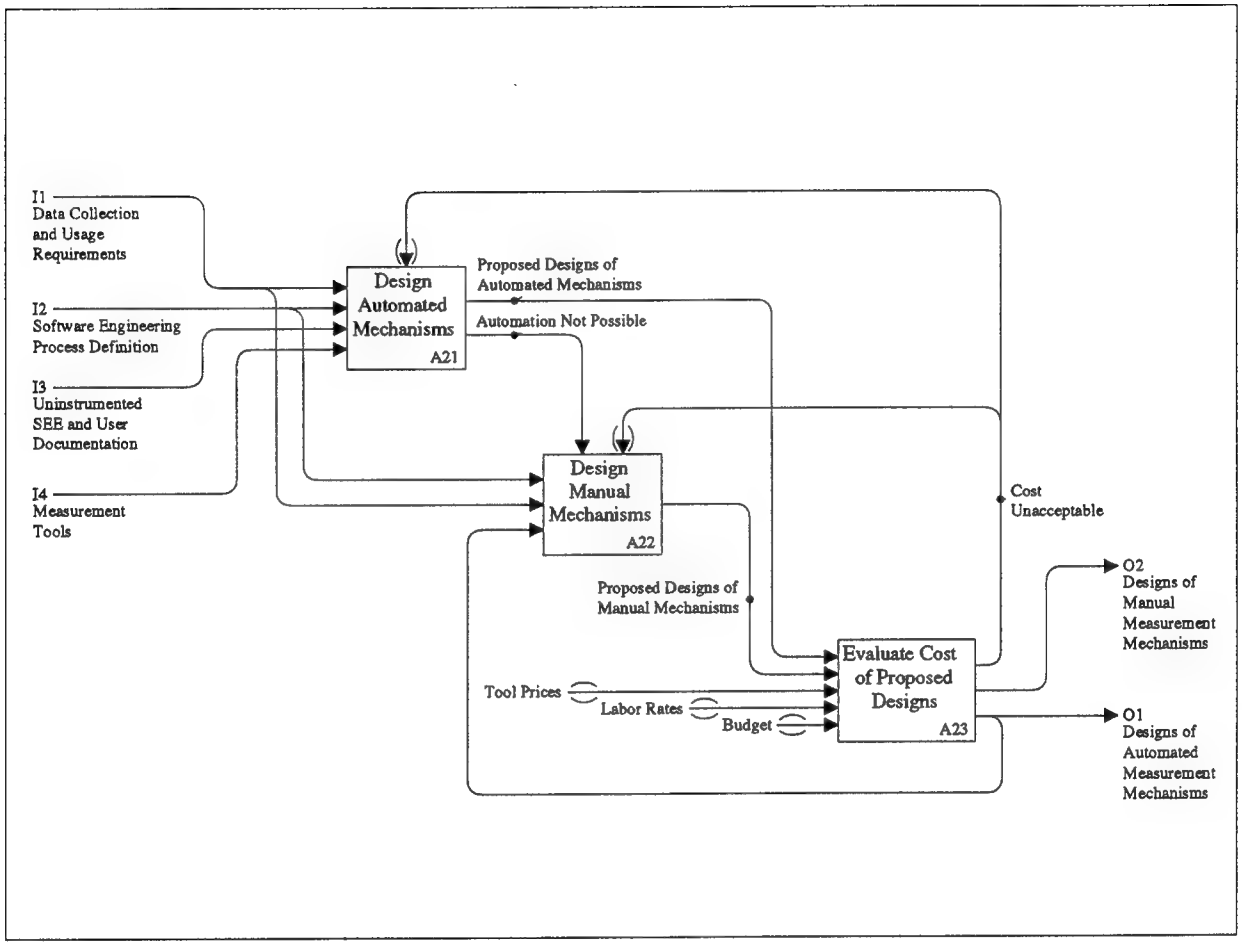



Figure 4: Decomposition of Design Measurement Mechanisms

The Design Automated Mechanisms activity is concerned with designing automated mechanisms for collection analysis, reporting and/or feedback of measurement data. These automated measurement mechanisms typically consist of automated tools and tool integration "glue" that integrates them with each other and with the SEE. To minimize the cost of implementing automated measurement mechanisms, existing measurement tools should be used, whenever possible. Of course, only tools that meet the *data collection and usage requirements* should be considered. Furthermore, they should fit well with the defined software engineering process (as represented by the *software engineering process definition*) and they should integrate easily with the SEE; otherwise it may be more cost-effective to implement new tools. When suitable existing tools cannot be found, new measurement tools must be developed to satisfy the remaining data collection and usage requirements. Unfortunately, some of these requirements simply may not be satisfiable through automation – at least, not

in a cost-effective manner. In that case, they must be satisfied through the use of manual mechanisms, as indicated by the *Automation Not Possible* ICOM in Figure ??. The output of the Design Automated Mechanisms activity is a set of *proposed designs of automated mechanisms*. These proposed designs are supplied as input to the Evaluate Cost of Proposed Designs activity, which determines whether the cost of implementation and sustained operation of the automated measurement mechanisms is acceptable. If the cost is unacceptable, the designs may need to be reworked, as indicated by the *Cost Unacceptable* ICOM.

The Design Manual Mechanisms activity is concerned with designing manual mechanisms for collection analysis, reporting and/or feedback of measurement data. These manual measurement mechanisms typically consist of data collection forms and/or manual procedures for collecting data, analyzing data, reporting data and/or analysis results and interpreting reports to determine what feedback (if any) is necessary. In designing manual mechanisms, the approved designs of automated mechanisms may need to be considered so that the automated and manual mechanisms can be coordinated. This is particularly true when automated data collection is combined with manual analysis, reporting and feedback, or vice versa. The output of the Design Manual Mechanisms activity is a set of *proposed designs of manual mechanisms*. Like the proposed designs of automated mechanisms, these proposed designs of manual mechanisms are supplied as input to the Evaluate Cost of Proposed Designs activity, which determines whether the cost of implementation and sustained operation of the manual measurement mechanisms is acceptable. If the cost is unacceptable, the designs may need to be reworked, as indicated by the *Cost Unacceptable* ICOM.

The Evaluate Cost of Proposed Designs activity is concerned with determining whether the costs associated with the proposed designs of automated mechanisms and the proposed designs of manual mechanisms are acceptable. These costs include not only the cost of implementing the mechanisms according to the proposed designs, but the cost of sustaining operation of the mechanisms as well. Tool prices and labor rates are needed for estimation of these costs. If the costs are acceptable, according to the budget, the designs are approved for implementation. If the costs are unacceptable, the design activities are repeated, to rework the designs. The outputs of the Evaluate Cost of Proposed Designs activity are the approved designs (both the designs of manual measurement mechanisms and the designs of automated measurement mechanisms). These become inputs for the Implement Measurement Mechanisms activity.

The sequencing of the activities, in this case, is not sequential. The Design Automated Mechanisms activity occurs first, reflecting the preference for automated measurement mechanisms over manual measurement mechanisms. If the data collection and usage requirements can be satisfied entirely by automated mechanisms, the Design Manual Mechanisms activity may not occur at all. This is usually not the case, however. When it is discovered that automation is not possible, the Design Manual Mechanisms activity is triggerred. Meanwhile, the Evaluate Cost of Proposed Designs activity may proceed for any proposed designs of automated mechanisms. As proposed designs of manual mechanisms are produced, the Evaluate Cost of Proposed Designs activity may proceed for those designs as well. A separate proposed design may be produced for each mechanism or several mechanisms may be grouped

together into a single proposed design; either way will work. Iteration of the Design Automated Mechanisms and/or Design Manual Mechanisms activities may be triggerred if the Evaluate Cost of Proposed Designs activity determines that the costs associated with the proposed designs are unacceptably high. Thus, it is quite possible for all three activities to be occurring simultaneously.

### 2.4.1 Node A21: Design Automated Mechanisms

Figure 5 presents the decomposition of the Design Automated Mechanisms activity. This activity consists of five subactivities: *Develop Tool-Independent Designs*, *Identify Tool Requirements*, *Select Measurement Tools*, *Design Additional Tools Needed* and *Design Tool Integration "Glue"*.

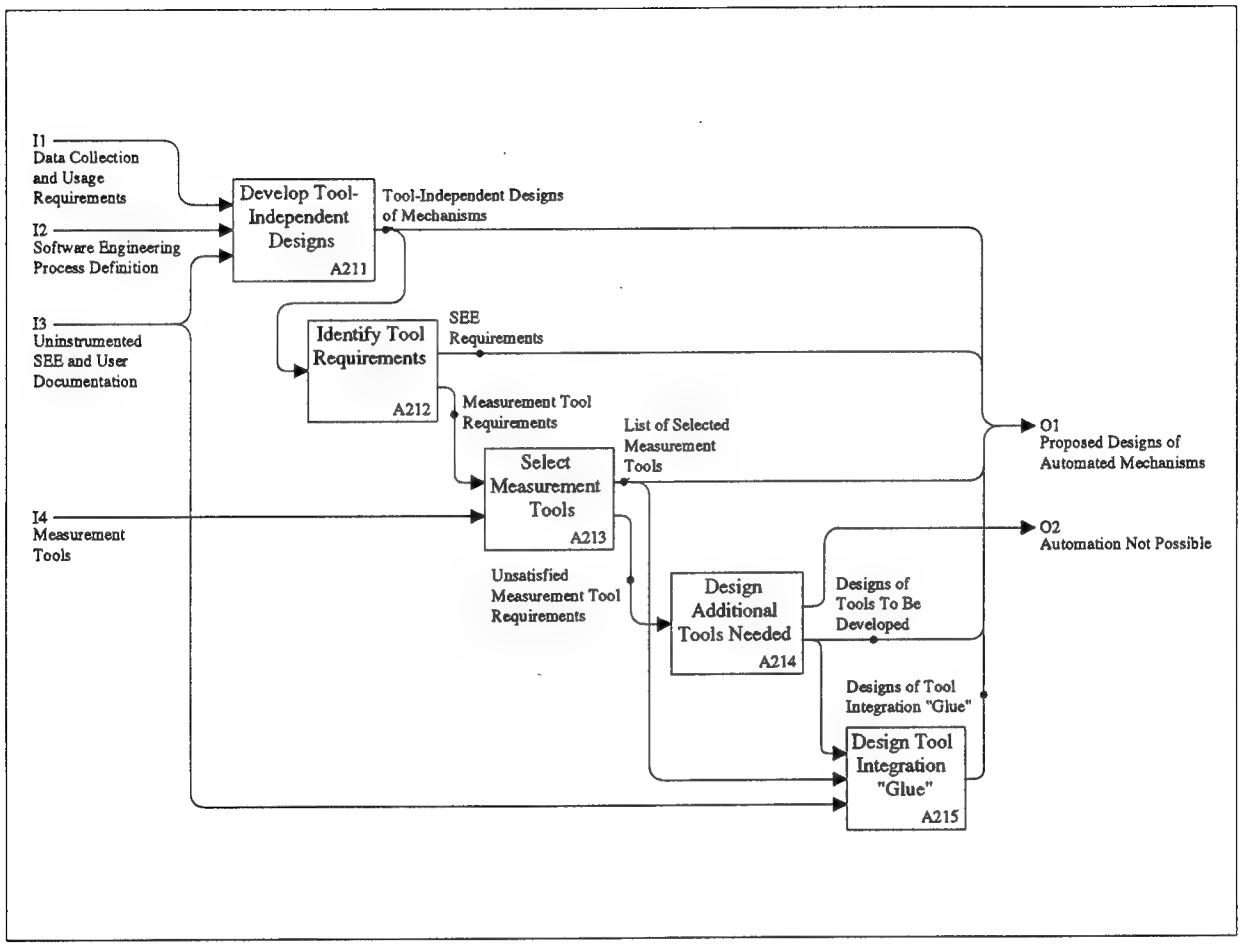



Figure 5: Decomposition of Design Automated Mechanisms

The Develop Tool-Independent Designs activity is concerned with designing mechanisms for collection, analysis, reporting and feedback of measurement data that are independent of the tools (if any) used to implement the mechanisms. The output of this activity consists of *tool-independent designs of mechanisms.* These are supplied as input to the Identify Tool Requirements activity (i.e., these designs are used as the basis for determining what tool capabilities and supporting SEE capabilities are needed to implement the measurement mechanisms). They also become a part of the proposed designs of automated mechanisms.

The Identify Tool Requirements activity is concerned with identifying requirements for automated tools that would implement the measurement mechanisms specified by the tool-indepent designs of mechanisms. These requirements include not only requirements for

automated measurement tools, but requirements for SEE capabilities that must be provided to support those tools as well. For example, if metrics concerning software problems are to be collected automatically, the SEE must provide a capability to track software problem reports on-line. The outputs of the Identify Tool Requirements activity consist of a set of *measurement tool requirements* and a set of *SEE requirements.* The measurement tool requirements are supplied as an input to the Select Measurement Tools activity. The SEE requirements become a part of the proposed designs of automated mechanisms. They are also an overall output of the Instrument Software Process activity.

The Select Measurement Tools activity is concerned with evaluating existing measurement tools (which may be COTS, public domain or proprietary) to determine which ones should be chosen to satisfy the measurement tool requirements. If not all of the measurement tool requirements can be satisfied using existing tools, a set of tools is chosen that together satisfy as many of these requirements as possible. The outputs of the Select Measurement Tools activity consists of a *list of the selected measurement tools* and a set of *unsatisfied measurement tool requirements.* The former is supplied as an input to the Design Tool Integration "Glue" activity and also becomes a part of the proposed designs of automated mechanisms. The latter is supplied as an input to the Design Additional Tools Needed activity.

The Design Additional Tools Needed activity is concerned with designing additional measurement tools to be developed to meet the unsatisfied measurement tool requirements. This is necessary in cases where not all measurement tool requirements can be satisfied by existing tools. It is also necessary in cases where the existing measurement tools are prohibitively expensive. In some cases, it may be determined that the unsatisfied measurement tool requirements simply are not satisfiable by automated means, as indicated by the *Automation Not Possible* ICOM. The output of the Design Additional Tools Needed activity consists of *designs of tools to be developed.* These are supplied as input to the Design Tool Integration "Glue" activity. They also become a part of the proposed designs of automated mechanisms.

The Design Tool Integration "Glue" activity is concerned with designing the software that needs to be developed to integrate measurement tools with each other and with the uninstrumented SEE. The output of this activity consists of *designs of tool integration "glue"* (i.e., designs of the integration software). These designs become a part of the proposed designs of automated mechanisms. The proposed designs of automated mechanisms subsequently become an input for the Evaluate Cost of Proposed Designs activity.

The dependencies between these activities impose a more or less sequential ordering. The Develop Tool-Independent Designs activity must occur before the Identify Tool Requirements activity, which in turn must occur before the Select Measurement Tools activity, which in turn must occur before either the Design Additional Tools Needed activity or the Design Tool Integration "Glue" activity. The Design Additional Tools Needed activity is optional; it need only be performed if not all of the measurement tool requirements can be satisfied by existing tools. If this activity is performed, it must occur before the Design Tool Integration "Glue" activity. Not all of the measurement mechanisms need be designed together, however.

Only mechanisms that support the same software metric (i.e., mechanisms for collection, analysis, reporting and feedback of the metric) need be designed together. The mechanisms for separate metrics can usually be designed separately. In this case, the entire sequence of activities can be iterated for each separate metric; or alternatively, these activities may be performed in a pipelined fashion.

### 2.4.2 Node A22: Design Manual Mechanisms

Figure 6 presents the decomposition of the Design Manual Mechanisms activity. This activity consists of two subactivities: *Design Data Collection Forms* and *Define Manual Procedures.*

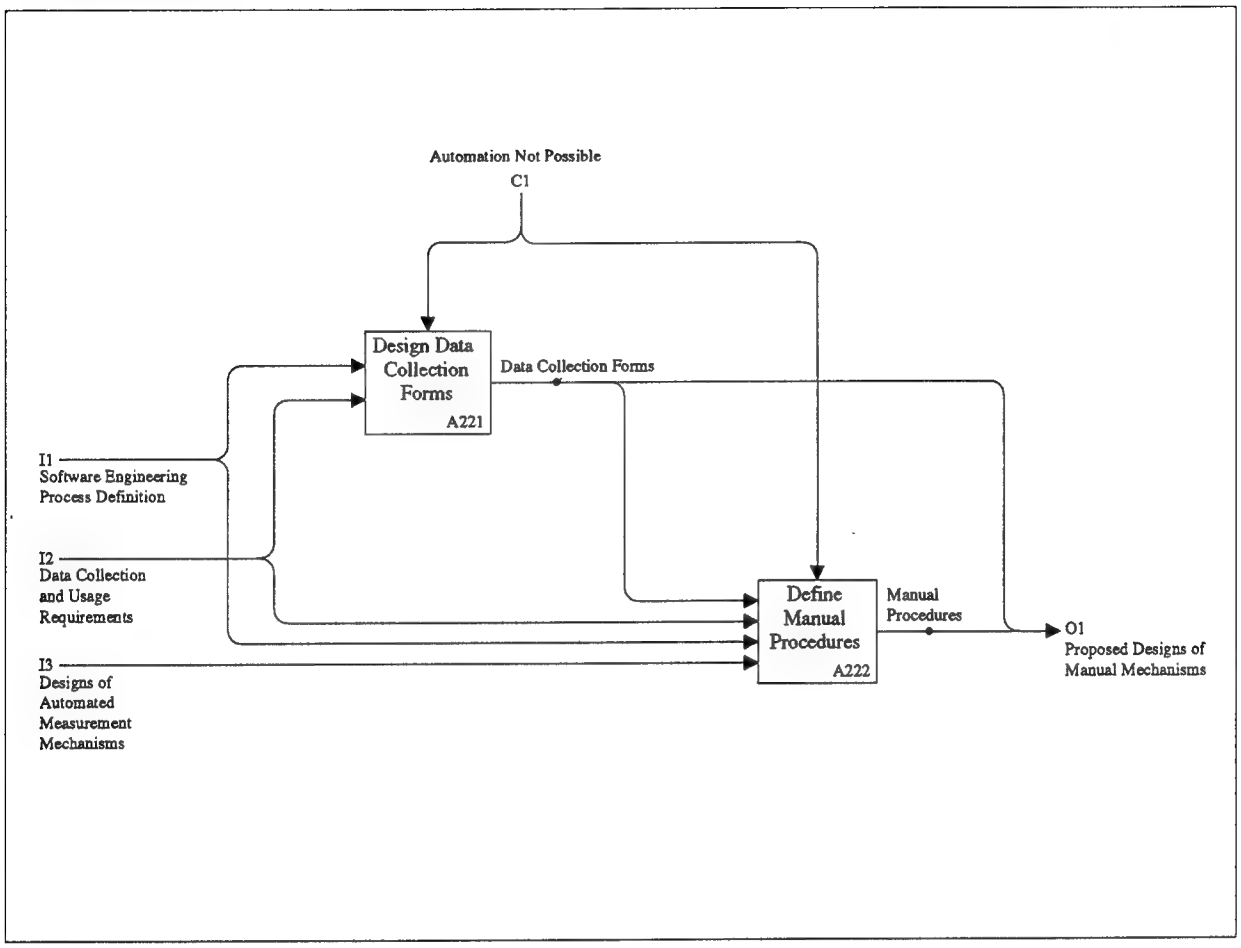



Figure 6: Decomposition of Design Manual Mechanisms

The Design Data Collection Forms activity is concerned with laying out the *data collection forms* that will be needed to obtain the required measurement data from human sources. The output of this activity consists of the layed out data collection forms. These forms are supplied as an input to the Define Manual Procedures activity. They also become a part of the proposed designs of manual mechanisms.

The Define Manual Procedures activity is concerned with defining *manual procedures* for analyzing measurement data, generating metric reports, and interpreting metric reports to determine what feedback (if any) is required for software engineering processes. This activity is only performed when there are data collection and usage requiremnts that cannot be satisfied by automation – at least, not in a cost-effective way. The Define Manual Procedures activity must consider the measurement data that are available from manual data collection forms, as well as the data that are available from automated data collection mechanisms. It

must also consider the designs of any automated measurement mechanisms that will be used for analysis, reporting and feedback of measurement data. The output of the Define Manual Procedures activity consists of the defined set of manual procedures. These become a part of the proposed designs of manual mechanisms. The proposed designs of manual mechanisms are subsequently supplied as an input for the Evaluate Cost of Proposed Designs activity.

To the extent that data collected manually are analyzed, reported and fed back manually, these activities must be performed sequentially. It is often the case, however, that manual analysis, reporting and feedback are combined with automated data collection, rather than manual data collection. And it is sometimes the case that manual data collection is combined with automated analysis, reporting and feedback. In these cases, the Design Data Collection Forms activity and the Define Manual Procedures activity can pe performed in parallel. Also, iteration of the sequence of activities is possible, because mechanisms to support separate metrics can be designed separately.

### 2.4.3 Node A23: Evaluate Cost of Proposed Designs

Figure 7 presents the decomposition of the Evaluate Cost of Proposed Designs activity. This activity consists of four subactivities: *Estimate Cost of Automated Mechanisms*, *Estimate Cost of Manual Mechanisms*, *Evaluate Cost* and *Approve Designs.*

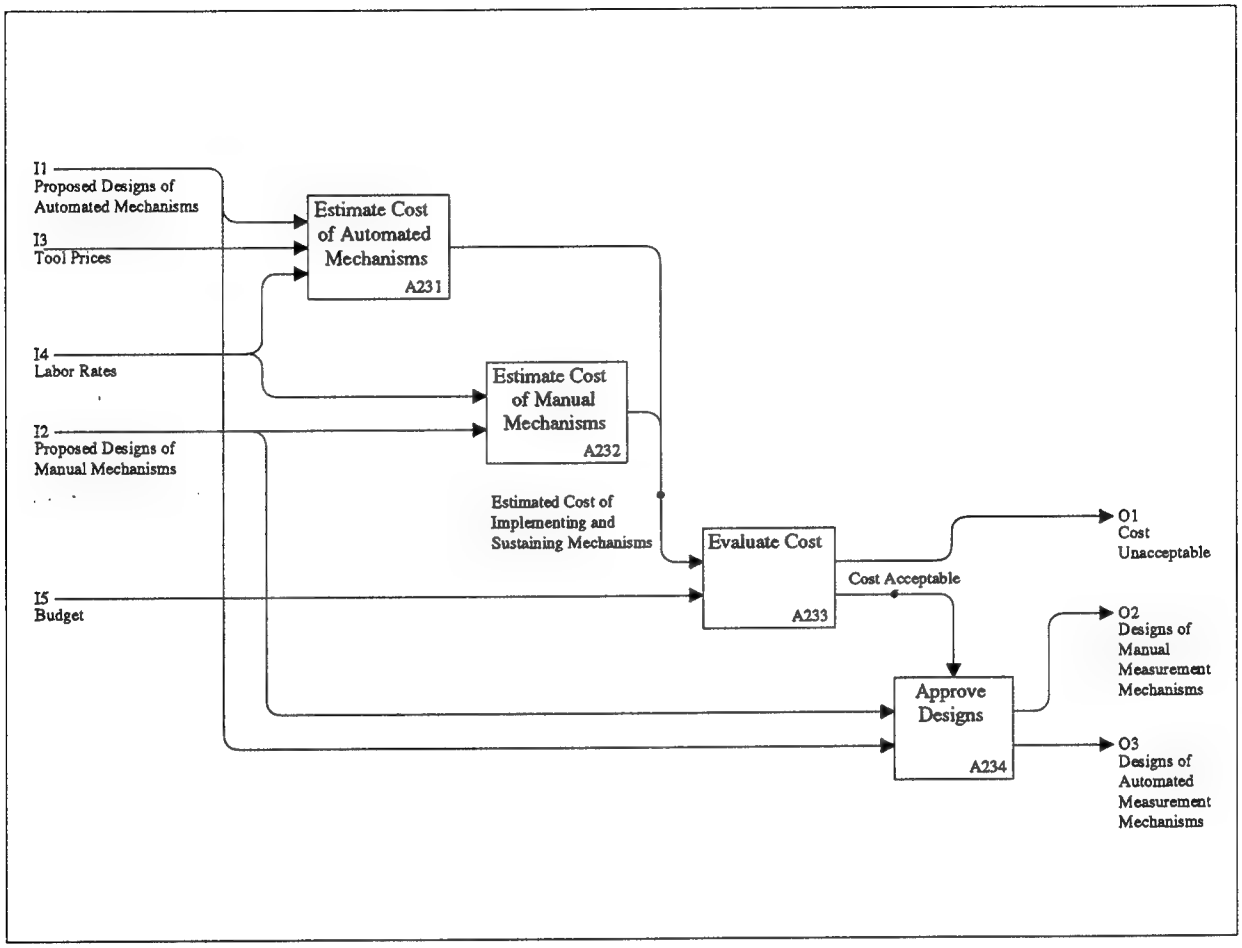



Figure 7: Decomposition of Evaluate Cost of Proposed Designs

The Estimate Cost of Automated Mechanisms activity is concerned with estimating both the cost of implementation and the cost of sustained operation of the proposed designs of automated mechanisms. The cost of implementation includes the cost of developing and/or purchasing the proposed measurement tools and the cost of integrating the tools with the SEE (and with each other, if necessary). The cost of sustained operation includes both the cost of human interaction with the tools (if any) and the cost of computer resources consumed by the tools. The output of the Estimate Cost of Automated Mechanisms consists of these cost estimates, which are supplied as input to the Evaluate Cost activity.

The Estimate Cost of Manual Mechanisms activity is concerned with estimating both the cost of implementation and the cost of sustained operation of the proposed designs of manual mechanisms. The cost of implementation includes the cost of publishing (and distributing)

the data collection forms and the manual procedures for data collection, analysis, reporting and feedback. The cost of sustained operation includes the cost of manually filling out the data collection forms, manually analyzing the collected data, manually producing metric reports and/or manually interpreting metric reports (and providing manual feedback by acting on those interpretations). The output of the Estimate Cost of Manual Mechanisms activity consists of these cost estimates, which are supplied as input to the Evaluate Cost activity.

The Evaluate Cost activity is concerned with determining whether the *estimated cost of implementing and sustaining mechanisms* for measurement is acceptable, according to the budget. If the cost is acceptable, the proposed designs of the measurement mechanisms may be approved for implementation (as indicated by the *Cost Acceptable* ICOM). If the cost is unacceptable, the proposed designs of the measurement mechanisms must be reworked (as indicated by the *Cost Unacceptable* ICOM). This activity has no outputs other than the cost acceptable and cost unacceptable indications, which are used to control the Approve Designs Design Automated Mechanisms and Design Manual Mechanisms activities.

The Approve Designs activity is concerned with approving designs of measurement mechanisms for implementation. To be approved, the estimated cost of implementation and sustained operation of the mechanisms must be deemed acceptable. The designs may be approved individually (i.e., for individual metrics) or as a group. The outputs of this activity are the approved designs of manual measurement mechanisms and the approved designs of automated measurement mechanisms. These outputs are supplied as inputs for the Implement Measurement Mechanisms activity.

Although the Estimate Cost of Automated Mechanisms and the Estimate Cost of Manual Mechanisms activities may be performed in parallel, the remaining activities must be performed sequentially – due to dependencies among them. Again, iteration of the sequence of activities is possible, because mechanisms to support separate metrics can be evaluated separately.

## 2.5 Node A3: Implement Measurement Mechanisms

Figure 8 presents the decomposition of the Implement Measurement Mechanisms activity. This activity consists of four subactivities: *Acquire Measurement Tools*, *Integrate Measurement Tools*, *Publish Data Collection Forms* and *Publish Manual Procedures.*

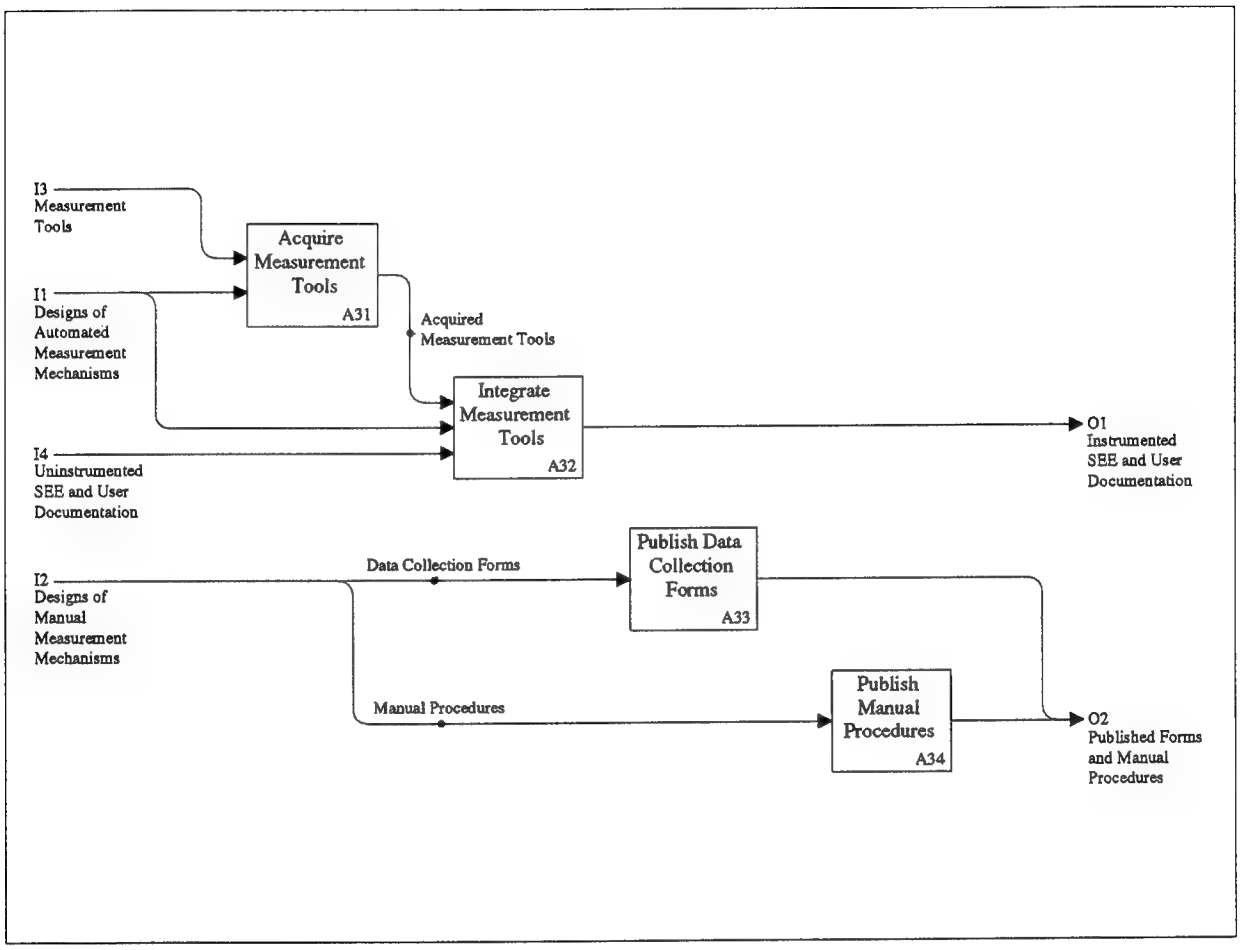



Figure 8: Decomposition of Implement Measurement Mechanisms

The Acquire Measurement Tools activity is concerned with purchasing and/or developing measurement tools as dictated by the approved designs of automated measurement mechanisms. The designs of automated measurement mechanisms include the list of existing measurement tools that have been selected for purchase. They also include the designs of the additional measurement tools to be developed. The outputs of the Acquire Measurement Tools activity are the actual tools that have been acquired (i.e., the *acquired measurement tools*). These are supplied as input to the Integrate Measurement Tools activity.

The Integrate Measurement Tools activity is concerned with integrating the acquired measurement tools with the uninstrumented SEE (and with each other), according to the approved designs of automated measurement mechanisms. It is also concerned with incorporate the user documentation for the acquired measurement tools into the user documentation for

the SEE. The designs of automated measurement mechanisms include the designs of the software necessary to integrate the acquired tools (i.e., the so-called tool integration "glue"). The output of the Integrate Measurement Tools activity is an instrumented SEE and its accompanying user documentation. This subsequently becomes a mechanism (in the ICOM sense of the word, mechanism) for the Execute Measurement Mechanism activity. Or more accurately, it becomes a resource to support the software engineering process. It is also an overall output of the Instrument Software Process activity.

The Publish Data Collection Forms activity is concerned with reproducing the approved data collection forms in quantity (so that they will be available for use in collecting measurement data). The output of the Publish Data Collection Forms activity is a supply of data collection forms. These are bundled together with published handbooks of manual procedures as a mechanism (in the ICOM sense of the word, mechanism) for the Execute Measurement Mechanism activity. Or more accurately, they become a resource to support the software engineering process. They are also an overall output of the Instrument Software Process activity.

The Publish Manual Procedures activity is concerned with publishing manual procedures. These published manual procedures take the form of handbooks containing instructions on how to manually collect, analyze, report and/or interpret and feed back measurement data. These handbooks of manual procedures are bundled together with data collection forms as a mechanism (in the ICOM sense of the word, mechanism) for the Execute Measurement Mechanism activity. Or more accurately, they become a resource to support the software engineering process. They are also an overall output of the Instrument Software Process activity.

There are two independent sets of activities represented here: the activities associated with implementing automated measurement mechanisms; and the activities associated with implementing manual measurement mechanisms. The former consists of the Acquire Measurement Tools and Integrate Measurement Tools activities, which must be performed sequentially (in that order). The latter consists of the Publish Data Collection Forms and Publish Manual Procedures activities, which may be performed in parallel. Because the two sets of activities are entirely independent, any activity from one set may be performed in parallel with any activity of the other. Iteration of these activities is also possible, because mechanisms to support separate metrics can be implemented separately. On the other hand, a more efficient implementation may be possible if some of the metrics are implemented together, since they may share common mechanisms (e.g., a single form may be used to collect data for multiple metrics, rather than a separate form for each).

### 2.5.1 Node A31: Acquire Measurement Tools

Figure 9 presents the decomposition of the Acquire Measurement Tools activity. This activity consists of two subactivities: *Purchase Measurement Tools* and *Develop Measurement Tools.*

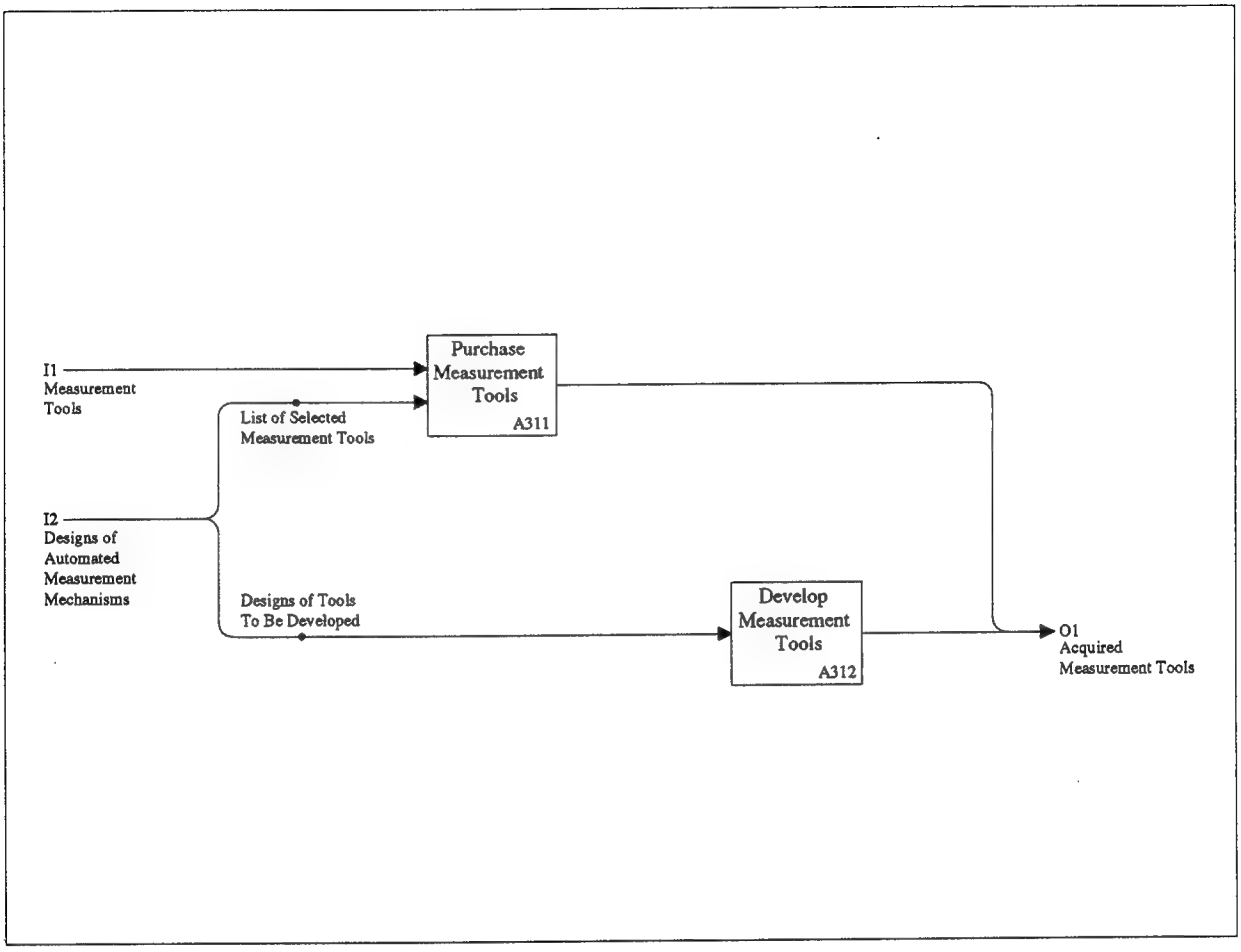



Figure 9: Decomposition of Acquire Measurement Tools

The Purchase Measurement Tools activity is concerned with purchasing the selected measurement tools, as specified by the list of selected measurement tools included in the design of automated measurement mechanisms. Some of these tools may (effectively) be purchased at no cost, if they are public domain or if they are owned by the company for which they are being acquired. The output of the Purchase Measurement Tools activity is the tools that have been purchased (and their associated user documentation). These are bundled with developed measurement tools and supplied as input to the Integrate Measurement Tools activity.

The Develop Measurement Tools activity is concerned with developing additional measurement tools, as specified by the tool designs included in the design of automated measurement mechanisms. The output of the Develop Measurement Tools activity is the tools that have been developed (and their associated user documentation). These are bundled with

purchased measurement tools and supplied as input to the Integrate Measurement Tools activity.

These two activities may be performed in parallel, since they are relatively independent.

## 2.6 Node A4: Execute Measurement Mechanisms

Figure 10 presents the decomposition of the Execute Measurement Mechanisms activity. This activity consists of four subactivities: *Execute Data Collection Mechanisms*, *Execute Analysis Mechanisms*, *Execute Reporting Mechanisms* and *Execute Feedback Mechanisms.*

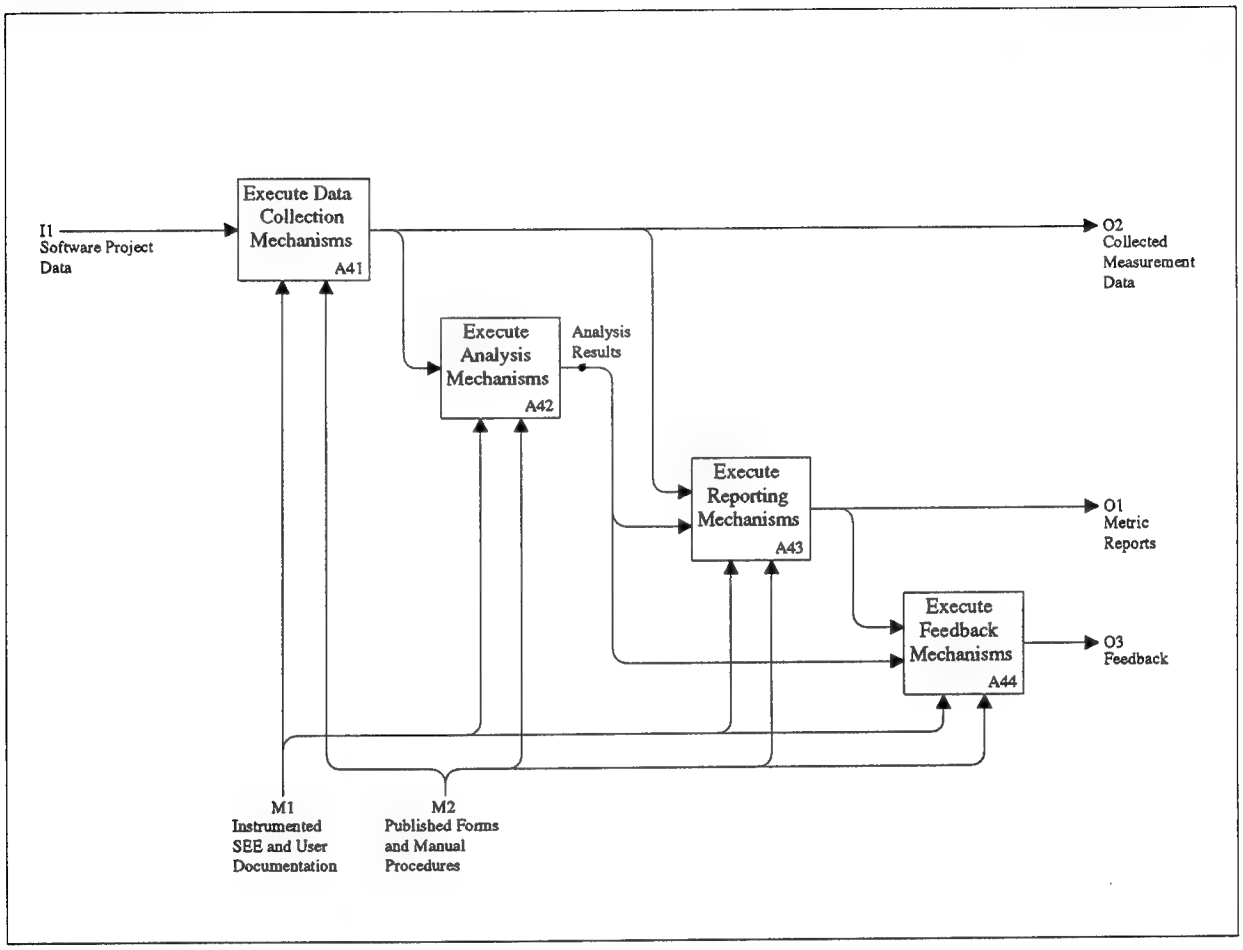



Figure 10: Decomposition of Execute Measurement Mechanisms

The Execute Data Collection Mechanisms activity is concerned with executing the automated and/or manual mechanisms that have been implemented to collect measurement data. These mechanisms typically collect the measurement data by monitoring and/or processing software project data. The output of the Execute Data Collection activity consists of the collected measurement data. This data is supplied as an input to either the Execute Analysis Mechanisms activity or the execute Reporting Mechanisms activity. It is also an overall output of the Instrument Software Process activity.

The Execute Analysis Mechanisms activity is concerned with executing the automated and/or manual mechanisms that have been implemented to analyze collected measurement data. The output of the Execute Analysis Mechanisms activity consists of *analysis results.* These are supplied as input to either the Execute Reporting Mechanisms activity or the Execute Feedback Mechanisms activity.

The Execute Reporting Mechanisms activity is concerned with executing the automated and/or manual mechanisms that have been implemented to generate metric reports from collected measurement data and analysis results. The output of the Execute Reporting Mechanisms activity consists of metric reports. These are supplied as input to the Execute Feedback Mechanisms activity. They are also an overall output of the Instrument Software Process activity.

The Execute Feedback Mechanisms activity is concerned with executing the automated and/or manual mechanisms that have been implemented to interpret analysis results and metric reports and produce feedback. Feedback can be used to control the software engineering process that is being measured. It may also serve as an input into the software process definition process (or the software process improvement process, if there is such a process), for use in software process improvement. The output of the Execute Feedback Mechanisms activity consists of feedback to be used for control the software engineering process. This feedback is an overall output of the Instrument Software Process activity.

These activities are partially ordered. The Execute Data Collection Mechanisms activity must be performed before the Execute Analysis Mechanisms or Execute Reporting Mechanisms activities can be performed. This is because the data must be gathered before it can be analyzed or reported. Similarly, the Execute Analysis Mechanisms or the Execute Reporting Mechanisms activity must be performed before the Execute Feedback Mechanisms activity can be performed. This is because the data must be analyzed first, if the analysis results are to be fed back automatically; and the analysis results must be reported first, if they are to be fed back manually.

# 3 Amadeus Guidelines

The preceeding section described the PIP in generic terms, as if it were totally independent of software measurement technology employed. This is because it treats the activities at a fairly high level of abstration. The fact is that, when it comes to applying the PIP in the context of a particular set of software measurement tools and techniques, the PIP may need some tailoring.

This section presents some specific tailoring guidelines for applying the PIP in the context of Amadeus. These guidelines are based on experience gained in using Amadeus and early prototypes of the PIP on a TRW project (supported by STARS) during the STARS U-Increment. Mostly, they concern the design and implementation of automated measurement mechanisms.

Before proceeding any further you may find it useful to review the capabilities of Amadeus. An overview of these capabilities is provided in Appendix A, for reference.

## 3.1 Determining Measurement Requirements

The *Determine Measurement Requirements* activity is not one that requires tailoring for application of various software measurement technologies. This activity involves determining what measurement data to collect and for what purpose, and when to collect it and when to make use of it. It does not involve determining what mechanisms will be used to collect, analyze, report or feedback the measurement data, or how to implement those mechanisms. These issues concerning mechanisms are the ones that are addressed differently when using different software measurement technologies. The tailoring required to deal with these differences appears in later activities.

## 3.2 Designing Measurement Mechanisms

Amadeus enables a higher degree of automation than some other sofware measurement technologies. In addition to automating collection of measurement data, it can automate analysis of measurement data, reporting of both the raw data and analysis results, and feedback of the analysis results into software engineering processes. Consequently, when Amadeus is used, fewer manual measurement mechanisms are designed and the automated measurement mechanisms that are designed are not limited to data collection.

Yet, there are always some situations where manual measurement mechanisms are required. This is because automation is not always possible; or if it is possible, it may not be cost-effective. For instance, plans or estimates (e.g., estimated lines of code for each software build) usually cannot be collected automatically. Similarly, measurements involving off-line activities (e.g., design walkthroughs) usually cannot be collected automatically. If this data is to be collected at all, it must be collected manually.

The analysis, reporting and feedback of this data, however, need not be performed manually. Amadeus provides utilities that make it possible to create data entry programs through which manually collected measurement data may be entered into the on-line measurement database. Once the data is the on-line database, automated mechanisms may be used for analysis, reporting and feedback.

Just as there are always some situations where automation is not possible or practical, there are also some situations where manual mechanisms are impractical. For example, it is usually not possible to collect computer resource utilization measurements manually; there is no way for a human to directly observe the resources being used. As another example, consider the classic problem of collecting source code size data. It is usually not cost-effective for a human to count lines of code manually.

Just as data that is collected manually may be analyzed, reported and fed back into the process automatically, if it is entered into the on-line database, data that is collected automatically may be analyzed, reported and fed back manually. Amadeus supports output of both raw measurement data and analysis results in human-readable format, so they may be analyzed and reported manually. It also supports automatic generation of metric reports (both tabular reports and graphical reports), from which feedback may be obtained through manual interpretation.

The fact that Amadeus emphasizes the use of automated mechanisms means that the costs of implementing measurement mechanisms will generally be higher than for other software measurement technologies where more manual mechanisms are used. This should be more than offset by the manual effort saved over the life of the project. If it is not, manual mechanisms should be used instead. Amadeus provides the flexibility to combine both automated and manual measurement mechanisms in the most cost-effective way. To ensure that automation is applied only where it will be cost-effective, the design of each proposed measurement mechanism should be carefully evaluated from the standpoint of cost (both the cost of implementing the mechanism and the cost of using it over the lifecycle of the project).

### 3.2.1 Designing Automated Mechanisms

The first two subactivities of the Design Automated Mechanisms activity, in the PIP, do not require any tailoring for Amadeus. The tool-independent designs of mechanisms are (by definition) independent of the measurement tools used, so the Develop Tool-Independent Designs activity ignores Amadeus just like it would ignore any other measurement tool. Similarly, the Identify Tool Requirements activity does not treat Amadeus differently from any other measurement tool, since it takes a tool-independent design as input.

The other three subactivities of the Design Automated Mechanisms activity do require some tailoring for Amadeus, however. The Select Measurement Tools activity should include Amadeus in the list of selected measurement tools. It should also include ease of integration with Amadeus as one of the criteria for selecting other measurement tools. This activity

should treat the built-in Amadeus agents just like other measurement tools; though these should be preferred over other comparable measurement tools, since they are already integrated with Amadeus and since they are essentially available at no cost (i.e., the cost is included in the price of Amadeus). The Design Additional Tools Needed activity should take the Amadeus application program interface (API) into account, so that the tools can be designed to take advantage of the capabilities provided by Amadeus and so that they can be easily integrated as Amadeus agents. The Design Tool Integration "Glue" activity should also make use of capabilities provided by Amadeus. Specifically, it should make use of Amadeus specifications and event generation capabilities.

The outputs of these three activities should include the following details, which are necessary for implementation of measurement tools and tool integration "glue":

- Identification of the measurement tools to be acquire and an indication as to whether they are to be purchased or developed;
- Detailed designs of any measurement tools to be developed;
- Identification of the Amadeus events to be generated and identification of the places in the SEE where the event generation calls are to be inserted;
- Designs of Amadeus specifications to trigger the agents in response to the events.

*More specific guidelines based on lessons learned are TBD. They will be provided in future drafts of this document.*

### 3.2.2 Designing Manual Mechanisms

As noted in Section 3.2, fewer manual measurement mechanisms are designed when Amadeus is used than when other software measurement technologies are used. This is mainly because Amadeus provides greater support for automation. One should remember this fact when designing manual measurement mechanisms. It may be that automation is possible and automated mechanisms would be more cost-effective. This is particularly likely if the manual mechanism is concerned with analysis of metric data or interpretation of metric reports. These are measurement tasks that are not normally automated and that Amadeus can automate quite effectively. The PIP, as it is defined in Section 2, is biased towards automation; so there is not much chance of missing an opportunity for automating measurement, if the PIP is followed literally. Nevertheless, one should be on the lookout for these unique opportunities for automation that are afforded by Amadeus.

### 3.2.3 Evaluating Cost of Proposed Designs

The costs of proposed designs of measurement mechanisms should be carefully evaluated regardless of whether Amadeus is used or some other software measurement technology is used. Consequently, this part of the PIP does not require any tailoring for Amadeus.

However, we have learned some lessons from TRW's experience with Amadeus that are useful to apply at this point in the PIP. In evaluating the cost of proposed designs of measurement mechanisms for the TRW project, we learned that most of the effort in developing Amadeus agents for a project is spent in developing data collection and feedback agents. These kinds of Amadeus agents typically require more effort to develop than other kinds of agents because they have to interface with other tools in the SEE. Reporting agents, on the other hand, typically require much less effort to develop, since they can usually be implemented entirely with Amadeus' own built-in capabilities (e.g., Amadeus_query, Amadeus_summary, Amadeus_format, Amadeus_graph). The effort to develop analysis agents varies according to the complexity of the analysis, but Amadeus' built-in capabilities can typically be exploited to some extent for analysis agents as well.

### 3.3 Implementing Measurement Mechanisms

To implement measurement mechanisms with Amadeus, one must perform three tasks:

- Instrument portions of the environment to generate events at key points in the processes being measured;
- Acquire (i.e., build or buy) agents to perform collection, analysis, reporting and/or feedback of measurement data at these key points; and
- Create Amadeus specifications to trigger the agents based on the events.

The first and the third task are part of the Integrate Measurement Tools activity. The second is part of the Acquire Measurement Tools activity. These are discussed further in the corresponding subsection, below.

The necessary information to accomplish these tasks should be available from the Design Measurement Mechanisms activity (specifically, from the Design Automated Mechanisms activity).

#### 3.3.1 Acquiring Measurement Tools

When using Amadeus, measurement tools are usually integrated as Amadeus agents – though they may sometimes be foreign (i.e., external) tools from which measurement data is imported or to which measurement data is exported. These tools may be acquired from any or all of the following sources:

- Agents bundled with the Amadeus commercial product (i.e., included in the purchase price);
- COTS tools (or public domain tools) from third parties;

- Development of measurement tools.

Acquisition of measurement tools from the first two sources corresponds to the Purchase Measurement Tools activity of the PIP. In fact, acquisition of Amadeus itself would logically be handled in this activity. Development of measurement tools corresponds to the Develop Measurement Tools activity of the PIP.

The measurement tools to be purchased or developed should be specified as part of the Design Measurement Mechanisms activity. For tools that are to be developed, it is important that a detailed (code-to) design of each tool be produced during the Design Measurement Mechanisms activity as well. Otherwise, substantial rework of the code may be required during implementation (due to poorly defined interfaces, use of improper algorithms, etc.).

Coding measurement tools that are to serve as Amadeus agents should be straightforward, if the detailed design has been done. They can be coded in any language (or combination of languages) for which an Amadeus API exists – presently Ada, C and C-shell script.

### 3.3.2 Integrating Measurement Tools

When using Amadeus, there are two aspects to integrating measurement tools:

1. Integrating the measurement tools with Amadeus (as Amadeus agents); and
2. Integrating the Amadeus agents with the environment.

The former is accomplished by using Amadeus specifications to invoke the tools and by using Amadeus utilities to import or export data from Amadeus' database. The latter is accomplished by instrumenting the SEE to generate Amadeus events and by creating Amadeus specifications that specify which tools are to be invoked in response to which events.

Typically, built-in Amadeus agents and measurement tools specifically developed as Amadeus agents are already integrated with Amadeus, so only third-party tools (e.g., COTS or public domain tools) need go through an integration activity to integrate them with Amadeus. Many third-party tools can be integrated as Amadeus agents – not just data collection agents, but analysis, reporting and feedback agents as well. For example, COTS PDL processors that collect complexity metrics can be integrated as Amadeus data collection agents and COTS graph generators and report writers can be integrated as reporting agents.

These tools are integrated with Amadeus through the use of Amadeus specifications and utilities provided by Amadeus for importing and exporting measurement data. Amadeus specifications can be set up to invoke the third-party tools under appropriate conditions. Data generated by these tools can be imported into Amadeus' database through the use of the

Amadeus_import command. Data required by these tools for analysis, reporting and/or feedback into software engineering processes can be extracted from Amadeus' database via the Amadeus_export, Amadeus_query, Amadeus_summary and/or Amadeus_format commands.

Third-party tools are most easily integrated with Amadeus if they provide programmatic interfaces – especially command line interfaces – and if their input and output data formats are well documented.

One way to instrument the SEE to generate Amadeus events is to use environment monitor processes that run in the background. Amadeus comes with some built-in environment monitors that can be used for this purpose. One is Amadeus_clock, which monitors the system clock and generates an Amadeus event once every minute, ten minutes, hour, day, week and month. Another is Amadeus_monitor, which monitors the files in a specified UNIX directory and generates an Amadeus event each time a file in the directory is accessed or modified.

Another way to instrument the SEE to generate Amadeus events is to embed Amadeus_event calls within shell scripts (as Amadeus_event commands) and/or programs (as Amadeus_event procedure calls). In this case, the events are generated as a side effect of executing the shell scripts and/or programs.

Yet another way to generate Amadeus events is to invoke Amadeus_event interactively through the command line interface.

Creating Amadeus specifications is straightforward. Each specification is contained in a short text file. One line of the text file identifies the name of the specification. Another line identifies the agent to be invoked. A third line identifies whether the process that triggers invocation of the agent should be suspended until the agent finishes executing. The rest of the lines specify the conditions under which the agent is to be invoked.

The Integrate Measurement Tools activity of the PIP is also concerned with installation and configuration of the measurement tools. Installation and configuration of third-party measurement tools will vary from one tool to the next. Installation and configuration of Amadeus, itself, is fairly easy. One need only load the contents of the distribution tape and edit a few configuration files, as explained in the installation instructions included with Amadeus.

### 3.4 Executing Measurement Mechanisms

The details of how to execute measurement mechanisms typically depend on the kind of mechanism (i.e., manual or automated) an how it is implemented. When Amadeus is used to implement software measurement, execution of all automated measurement mechanisms is controlled through a common user interface, and in most cases, the mechanisms may be set up to execute automatically. This is typically not the case when using software measurement tools other than Amadeus; such tools usually must be executed manually and they tend to

have no common user interface.

Executing automated measurement mechanisms implemented with Amadeus involves running the Amadeus measurement framework and activating a set of Amadeus specifications that tell the Amadeus Interpreter which agents to invoke under what conditions (and how to invoke them).

Running the Amadeus measurement framework enables Amadeus to monitor events occurring within the environment and invoke Amadeus agents in response to those events. The Amadeus measurement framework consists of a set of background processes. It includes the Amadeus Interpreter (Amadeus_interpreter) and the clock monitor (Amadeus_clock). It may also include file system monitor processes (Amadeus_monitor) monitoring various parts of the file system.

Amadeus specifications can be activated by the user either from the GUI or from the command line interface. Alternatively, they may be activiated automatically via Amadeus' API. Amadeus' graphical user interface (GUI) consists of a control panel (an X window) through which users may control the Amadeus measurement framework.

The procedures for manually interpreting outputs produced by Amadeus agents should have been defined during the Design Measurement Mechanisms activity (or more specifically, during the Design Manual Mechanisms activity). They should have been published during the Implement Measurement Mechanisms activity (or mor specifically, during the Publish Manual Procedures activity). Whenever possible, interpretation of the collected measurement data should be automated – to avoid misinterpretation. Nevertheless, it may be necessary to interpret some of the data manually. In that case, the published manual procedures should be consulted.

## A Amadeus Overview

Amadeus is a software product that automates measurement activities within a software engineering environment. It provides services to facilitate the collection, analysis and reporting of software metrics – not just static measurements of software products, such as lines of code or pages of documentation, but measurements of quality factors and process characteristics as well. Amadeus also provides mechanisms for feedback of metrics into software engineering processes, to actively support empirical guidance and continuous process improvement (CPI).

The potential benefits of Amadeus include better visibility into software engineering processes, increased confidence in management and technical decisions, improved cost/schedule performance of software engineering organizations, and enhanced quality of software products.

By virtue of its open architecture, Amadeus is extremely flexible. It may be easily adapted to a wide variety of processes, implementation languages and measurement paradigms. It may also be extended to collect, analyze, report and feed back new metrics.

As shown in Figure 11, Amadeus consists of multiple software components. At the heart of Amadeus is a component called the Amadeus Interpreter. This component monitors significant events occurring within the software engineering environment (e.g., tool invocations, changes in source files or documents, creation of software problem reports, expiration of time intervals, even changes in collected measurement data) and invokes other components, called agents, to respond to the events. Some of these agents perform collection of measurement data (e.g., unobtrusively counting the number of lines of code in a source file, or prompting a user for some data that cannot be obtained unobtrusively). Others analyze the collected measurement data (e.g., building a predictive model based on statistical correlation of the data). Yet others produce visualizations of the measurement data (e.g., displaying graphs or tables of metrics) or use the data for feedback into the development process. Amadeus comes with a set of basic data collection, analysis and visualization agents. Users may extend the set of agents with their own, user-defined, agents.

Measurement data is collected in a historical database managed by Amadeus. This database is organized as sequences of data records, each record representing a particular occurrence of an event or a particular measurement of a metric value. Each data record consists of a predefined collection of data fields, including the date and time at which the event occurred or the metric value was measured, the name of the event or metric, the measured value of the metric, an indication of the source of the metric value or event (e.g., the name of the source file for a lines-of-code metric) and other contextual information that can later be used to aggregate the data. Amadeus provides capabilities to import data from other tools, export data to other tools, or calculate various statistics on the metric data (e.g., averages or totals) for use in analyses or visualizations. The import/export facility can be used, for example, to transfer the data to and from various spreadsheet programs.

Users interact with Amadeus through an OSF/Motif-based graphical user interface (GUI).

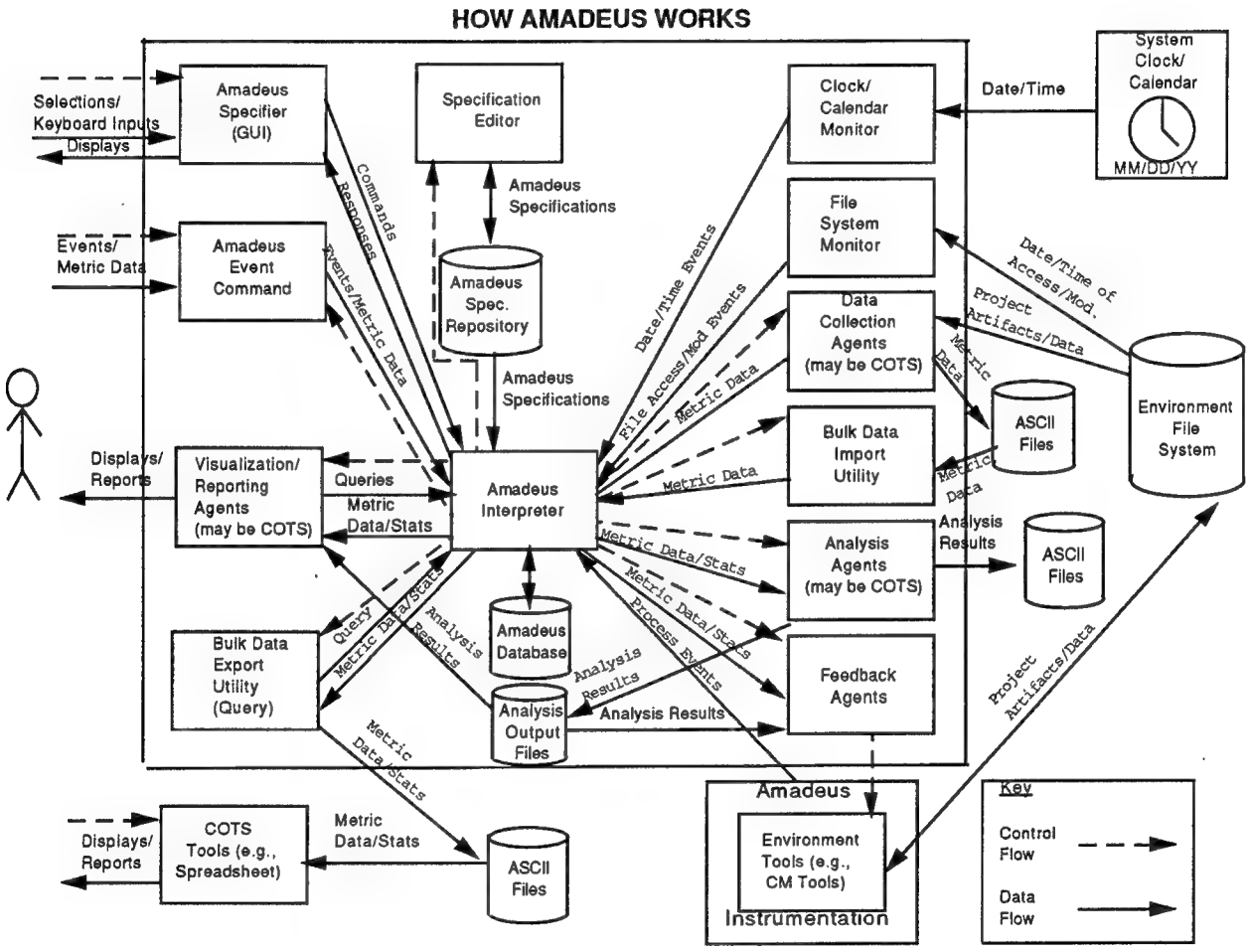



Figure 11: Amadeus Control and Data Flow

This interface provides users with the capability to dynamically control the dispatching of agents in response to events within the software engineering environment. In addition, it provides users with the capability to dynamically request hardcopy reports or activate and deactivate various on-screen displays.

Besides the graphical user interface, Amadeus provides an application program interface (API). This API allows users to create their own agents or to integrate other commercial off-the-shelf (COTS) measurement tools as agents. It also allows them to invoke the capabilities of the GUI from within programs or from a UNIX shell.

The following subsections briefly describe the various capabilities of Amadeus. These capabilities may be invoked either from the GUI or the API. When invoked from the GUI, they are invoked by using the mouse to select menu items and buttons from graphical displays. When invoked from the API, they are invoked either as commands issued from a UNIX shell script or as procedure calls issued from a program written in a language such as C. These capabilities may also be invoked interactively through a UNIX shell, thereby providing an alternative, command-line-based, user interface.

The following subsections sometimes mention specific Amadeus commands. Full descriptions of these commands can be found in the user documentation that accompanies the Amadeus commercial product.

### A.1 Data Collection

Measurement data may be collected by Amadeus in any combination of three ways:

1. Fully automatic data collection from the environment;
2. Interactive data entry; or
3. Importation of data from external sources.

In the first case, data collection agents unobtrusively collect the data either by scanning and analyzing the outputs of other tools in the environment or by communicating directly with those other tools (via their APIs). These agents are typically invoked automatically by Amadeus in response to events occurring within the environment.

In the second case, users supply the measurement data interactively. This may be achieved either through a data entry utility supplied with Amadeus (Amadeus_event) or through user-supplied data collection agents that implement on-line forms and questionnaires. Such data collection agents may be invoked automatically by Amadeus, in response to events occurring within the environment, or they may be invoked manually by the user.

In the third case, the measurement data must be extracted from external sources before it can be imported into Amadeus. For example, the data may be located on some other

machine that does not support Amadeus or it may reside in a proprietary database that is inaccessible to Amadeus. If the data can be extracted in ASCII format, it may subsequently be imported into Amadeus' database through a data import utility supplied with Amadeus (Amadeus_import). Depending on the available facilities for communicating with the external source of the data, this may or may not require manual intervention. If a programmatic interface is available for extracting the data from the external source, a data collection agent may be invoked automatically by Amadeus to extract the data and subsequently import it into Amadeus' database. If no such interface is available, a user may have to manually extract the data from the external source and either manually invoke Amadeus' data import utility or place the data somewhere where a data collection agent can get at it.

### A.1.1 Monitoring The Environment

Amadeus provides built-in utilities (Amadeus_clock and Amadeus_monitor) for monitoring two kinds of events occurring within the environment:

- Date/time events (from the system clock); and
- File access or modification events (from the file system).

In addition, the data entry utility supplied with Amadeus (Amadeus_event) also supports event generation, allowing users to create their own event monitors for other kinds of events (e.g., tool invocations).

### A.1.2 Triggering Agents

Amadeus specifications indicate which agents are to be invoked under which conditions. The conditions are based on information in Amadeus' database, which includes events and metrics values. Specifications may be activated via the Amadeus_activate command and deactivated via the Amadeus_deactivate command. A list of the available specifications may be obtained via the Amadeus_browse command. An individual specification may be displayed via the Amadeus_show command. The list of currently active specifications may be displayed via the Amadeus_show_active command. The syntax of Amadeus specifications is described in the user documentation that accompanies the Amadeus commercial product.

### A.1.3 Entering Data Interactively

Amadeus events may be generated interactively via the Amadeus_event command. Metric data may also be entered interactively via the Amadeus_event command.

### A.1.4 Importing Data from Foreign Tools

Metrics data generated by foreign tools (i.e., tools not included with the Amadeus release) may be imported into Amadeus' database via the Amadeus_import command. This facility can be used, for example, to record complexity measures reported by a PDL processor.

This same facility may be used to import data from external sources. For example, it may be used to import cost data downloaded from a cost accounting system running on another platform.

## A.2 Metrics Visualization and Reporting

Amadeus comes with utilities that support visualization and reporting of metrics. These utilities break down the task of metrics visualization and reporting into three steps:

1. Extracting the data from Amadeus' database;
2. Formatting the data in tabular form; and
3. Generating a graph of the data, if desired.

To apply the utilities, however, it is best to work backwards. First determine what data is necessary to produce the desired graph. Then determine how this data can be obtained by aggregating and formatting the data available in Amadeus' database. Lastly, define the queries that are necessary to extract the data from Amadeus' database. In some cases, you may find that additional data is needed that has not been collected; in such cases, the data collection agents will require modification.

### A.2.1 Extracting Data from Amadeus' Database

Measurement data may be extracted from Amadeus' database via the Amadeus_query command. This command extracts a sequence of data records that match a specified set of field values.

### A.2.2 Formatting The Data

Once extracted from Amadeus' database, the measurement data may be aggregated via the Amadeus_summary command. This command generates a table of the aggregated metric values, where the rows represent distinct values (or combinations of values) of certain key data fields (e.g., artifact name, month) and the columns represent aggregated values of various metrics (e.g., current lines of code, total defects reported).

Arithmetic may be performed on the aggregated metric values via the Amadeus_format command. For example, if one has aggregated the number of lines of code for an artifact in one column and the number of defects reported for that artifact in a second column, one can use Amadeus_format to calculate defects per thousand lines of code (i.e., by dividing the first column by 1000 and then dividing by the second column).

### A.2.3 Generating Graphs

Metric data in a tabular form may be displayed as a graph via the Amadeus_graph command. This command currently supports two kinds of graphs: line graphs and bar graphs. Line graphs may display multiple lines, representing multiple data sets (each taken from a separate column of the tabular input). Line graphs may also (optionally) have the individual data points highlighted. Bar graphs display only a single data set.

## A.3 Exporting Data to Foreign Tools

The same facilities that are used to generate tabular reports of the measurement data can be used to export the data to foreign tools (e.g., COTS spreadsheet programs). The data can be extracted from Amadeus' database via the Amadeus_query command and can then be aggregated and formatted for export via Amadeus_summary and Amadeus_format.

## A.4 Metrics Analysis/Integration

Unlike other metric tools, Amadeus automates not only collection and reporting of measurement data, but analysis of the data as well. It also allows multiple kinds of metrics to be integrated within the same analysis.

### A.4.1 Classification Analysis

One automated analysis technique supported by Amadeus is Classification Analysis. This analysis technique involves construction of a predictive model based on historical metric data. The model, which is called a classification tree, is a tree structure that integrates multiple kinds of metrics. It predicts a particular property of a particular kind of software artifact (e.g., error-proneness of a source code artifact) by indicating which values of which metrics are or are not characteristic of artifact instances having the property. This tree then serves as a decision tree for making predictions about similar artifact instances that are under development.

Amadeus supports Classification Analysis by providing two utilities: one to construct classification trees based on historical data (Amadeus_gen_tree_view); and one to apply the trees to predict properties of artifacts under development(Amadeus_apply_tree).

### A.5 Systematic Feedback and Empirical Guidance

Amadeus also supports automatic feedback of analysis results into ongoing software engineering processes, to empirically guide them. For example, analyses (such as Classification Analysis) of empirical measurement data may be used to identify software components that are likely to have defects, and feedback of those analysis results into the testing process may focus testing on the identified components. Thus, empirical data is used to guide the process such that resources are allocated where they are likely to provide the highest payoff.

There are basically two ways to implement feedback with Amadeus:

1. Have a shell script or a program query the metrics database at predefined points in the process and dynamically modify the process based on the metrics data (i.e., a programmed approach);
2. Activate an Amadeus specification to trigger additional or alternative process fragments in response to Amadeus events based on the metrics data (i.e., an event-driven approach).

# B Glossary

This Glossary is divided into three subsections, for ease of reference. The first subsection (B.1) defines terms that are used to discuss the IDEF0 process modeling paradigm. The second subsection (B.2) defines terms that are used to discuss the PIP. The third subsection (B.3) defines terms that are used to discuss Amadeus.

## B.1 IDEF0 Terminology

The following terms are used to discuss IDEF0 process models in general.

**Activity** – An action (which may be decomposed into subactions, or subactivities) that transforms a set of *inputs* into a set of *outputs*, possibly through the use of some *mechanisms*, and as governed by one or more *controls*; every activity has a name and a *node number* associated with it; in an IDEF0 diagram, each activity is represented by a box containing the name and the node number.

**Context Page** – A page of an IDEF0 model that corresponds to the top of the *activity* decomposition hierarchy; it contains a single activity box representing the overall process being modeled; the *ICOMs* attached to this activity box represent the context within which the process is assumed to operate.

**Control** – Something that governs when and/or how an *activity* is performed; in an IDEF0 diagram, a control for an activity is represented by an arrow attached (by its head) to the top side of the activity box.

**Decomposition Page** – a page of an IDEF0 model that depicts the decomposition of an *activity* into its subactivities and their *ICOMs* (the decomposed activity is referred to as the *parent activity*); an IDEF0 diagram will typically have many of these pages, in order to hierarchically decompose a process into several levels of activities.

**ICOM** – An *input*, *control* or *mechanism* feeding into an *activity* or an *output* coming out of an activity; in an IDEF0 diagram, each ICOM is represented by an arrow that has a label associated with it; branching of an ICOM arrow into multiple arrows typically indicates either unbundling of the ICOM into its constituent parts or alternative variants or distribution of the entire ICOM to multiple activities (in the former case, the individual branches are labeled; in the latter case, they are not); joining of multiple ICOM arrows into a single arrow typically represents either bundling of multiple ICOMS or multiple variants of an ICOM into a composite ICOM or collection of a single ICOM from multiple activities (again, the branches are labeled in the former case and not labeled in the latter case).

**Input** – A resource (e.g., some material or information) that is consumed by an *activity*; in an IDEF0 diagram, an input for an activity is represented by an arrow attached (by its head) to the left side of the activity box.

**Mechanism** – An entity (e.g., a person, a machine, a software tool) that performs an *activity* or is used to perform an activity without being consumed by the activity; in an IDEF0 diagram, a mechanism for an activity is represented by an arrow attached (by its head) to the bottom side of the activity box.

**Node Number** – A unique identifier associated with each *activity* in an IDEF0 diagram; in an IDEF0 diagram, this number is usually depicted in the lower right-hand corner of the activity box.

**Output** – A product or result coming out of an *activity*; in an IDEF0 diagram, an output for an activity is represented by an arrow attached (by its tail) to the right side of the activity box.

**Parent Activity** When applied to a *decomposition page*, refers to the *activity* whose decomposition is depicted on the decomposition page.

**Port** – A point of attachment of an *ICOM* to an *activity*; ports are classified as either *input* ports, *output* ports, *control* ports or *mechanism* ports, depending on which side of the activity box they are located on; on an IDEF0 *decomposition page*, the ports of the *parent activity* are represented by composite labels (typically around the edges of the page) consisting of a port number (of the form In, Cn, On or Mn, where n is an integer) indicating the position of the attachment point on the parent activity box and the label associated with the attached ICOM.

**Tunnel** – A notational convention for reducing clutter on an IDEF0 diagram; a tunnel may be applied to any *port* (or combination of ports) and has the effect of making the *ICOM* attached to the port invisible on the other side of the port; when applied to the ports around the sides of an *activity* box, tunnels make the ICOMs invisible on the *decomposition page* for the activity (it is assumed that they are still there, but that they are not significant enough to the activity to be worth cluttering the diagram); when applied to the ports around the edges of a decomposition page, they make the ICOMs invisible on the parent page (and in fact, they indicate that the ICOMs may actually come from somewhere other than the *parent activity*); in an IDEF0 diagram, a tunnel is represented by a set of parentheses surrounding the head or tail of an ICOM arrow.

## B.2 PIP Process Model Terminology

The specific terms that are used to discuss the PIP model fall into two categories: names of activities; and labels associated with ICOMs. For ease of reference, we have divided the following glossary of PIP terms into two parts, according to these categories.

**Activities:**

**Acquire Measurement Tools (A31)** – Purchase and/or develop *measurement tools* as dictated by the *designs of automated measurement mechanisms.*

**Approve Designs (A234)** – Approve those designs of measurement mechanisms (i.e., em designs of automated measurement mechanisms and *designs of manual measurement mechanisms*) for which the implementation and sustained operating costs have been deemed acceptable; the designs may be approved individually (i.e., for individual metrics) or as a group; once approved, the designs become inputs to the *Implement Measurement Mechanisms* activity.

**Define Manual Procedures (A222)** – In cases where automation is not possible, define *manual procedures* for analyzing collected measurement data, generating metric reports, and interpreting metric reports to determine what feedback (if any) is required for software engineering processes; this activity must consider the measurement data that are available from manual *data collection forms*, as well as the data that are available from automated data collection mechanisms; it must also consider the *designs of automated measurement mechanisms* that will be used for analysis, reporting and feedback of measurement data.

**Design Additional Tools Needed (A214)** – In cases where not all *measurement tool requirements* can be satisfied by existing *measurement tools* (commercial tools, public domain tools or otherwise), or where the existing measurement tools are prohibitively expensive, design additional measurement tools to meet the *unsatisfied measurement tool requirements*; in some cases, it may be determined that no automated tools can possibly satisfy the unsatisfied measurement tool requirements, so manual mechanisms may have to be used instead.

**Design Automated Mechanisms (A21)** – Design automated mechanisms, if possible, for collection, analysis, reporting and/or feedback of measurement data; automated measurement mechanisms typically consist of automated tools and tool integration "glue" that integrates them with each other and with the SEE; use existing measurement tools, if any can be found that meet the *data collection and usage requirements* and that fit well with the defined software engineering process (as represented by the *software engineering process definition*) and integrate easily enough with the uninstrumented SEE; otherwise design new measurement tools to be developed; the *proposed designs of automated mechanisms* may need to be reworked to reduce the total cost of implementation and sustained operation.

**Design Data Collection Forms (A221)** – Lay out the forms (i.e., *data collection forms*) that will be needed to obtain the required measurement data from human sources.

**Design Manual Mechanisms (A22)** – Where automation is not possible for collection, analysis, reporting and/or feedback of measurement data, design manual mechanisms; manual measurement mechanisms typically consist of *data collection forms* and/or *manual procedures* for collecting data, analyzing data, reporting data and/or analysis results and interpreting reports to determine what feedback (if any) is necessary; the *proposed designs of manual mechanisms* may need to be reworked to reduce the total cost of implementation and sustained operation.

**Design Measurement Mechanisms (A2)** – Design a cost-effective combination of automated and manual mechanisms to satisfy software measurement requirements (i.e., the *data collection and usage requirements* produced by the *Determine Measurement Requirements* activity); prefer to use automated mechanisms, if possible.

**Design Tool Integration "Glue" (A215)** – Design the software that needs to be developed to integrate measurement tools with each other and with the uninstrumented SEE to implement software measurement.

**Determine Measurement Requirements (A1)** – Determine what software metrics to collect and for what purpose; also, determine what individual pieces of information need to be collected, when they need to be collected, how they are to be used (e.g., to calculate the metrics, or to aggregate the data for analysis and/or reporting), and when they are to be used.

**Develop Measurement Tools (A312)** – Develop additional measurement tools, as specified by the *designs of tools to be developed*, to implement automated measurement mechanisms.

**Develop Tool-Independent Designs (A211)** – Design mechanisms for collection, analysis, reporting and feedback of measurement data, and do so in a manner that is independent of the tools (if any) used to implement the mechanisms.

**Estimate Cost of Automated Mechanisms (A231)** – Estimate both the cost of implementation and the cost of sustained operation of the *proposed designs of automated mechanisms*; the cost of implementation includes the cost of developing and/or purchasing the proposed measurement tools and the cost of integrating the tools with the SEE (and with each other, if necessary); the cost of sustained operation includes both the cost of human interaction with the tools (if any) and the cost of computer resources consumed by the tools.

**Estimate Cost of Manual Mechanisms (A232)** – Estimate both the cost of implementation and the cost of sustained operation of the *proposed designs of manual mechanisms*; the cost of implementation includes the cost of publishing (and distributing) the *data collection forms* and the *manual procedures* for data collection, analysis, reporting and feedback; the cost of sustained operation includes the cost of manually filling out the data collection forms, manually analyzing the collected data, manually producing metric reports and/or manually interpreting metric reports (and providing manual feedback by acting on those interpretations).

**Evaluate Cost (A233)** – Determine whether the *estimated cost of implementing and sustaining mechanisms* for measurement is acceptable, based on the *budget*; if so, output *cost acceptable* (to trigger approval of the proposed designs of the measurement mechanisms); otherwise, output *cost unacceptable.*

**Evaluate Cost of Proposed Designs (A23)** – Determine whether the costs associated with the *proposed designs of automated mechanisms* and the *proposed designs of manual*

*mechanisms* are acceptable; if they are acceptable, approve the designs; if they are not acceptable, output *cost unacceptable* (to trigger rework of the proposed designs).

**Execute Data Collection Mechanisms (A41)** – Execute the automated and/or manual mechanisms that have been implemented to collect measurement data by monitoring and/or processing *software project data.*

**Execute Analysis Mechanisms (A42)** – Execute the automated and/or manual mechanisms that have been implemented to analyze *collected measurement data.*

**Execute Reporting Mechanisms (A43)** – Execute the automated and/or manual mechanisms that have been implemented to generate *metric reports* from *collected measurement data* and *analysis results.*

**Execute Feedback Mechanisms (A44)** – Execute the automated and/or manual mechanisms that have been implemented to interpret *analysis results* and *metric reports* and produce *feedback.*

**Execute Measurement Mechanisms (A4)** – Execute the automated and/or manual measurement mechanisms that have been implemented to collect measurement data, analyze the *collected measurement data*, generate *metric reports* and/or produce *feedback*; the automated measurement mechanisms have been implemented by incorporating them into the *instrumented SEE and user documentation*; the manual measurement mechanisms have been implemented as *published forms and manual procedures.*

**Identify Data Collection & Usage Points (A13)** – Identify the points in the software engineering process (as defined by the *software engineering process definition*) where measurement data (as specified by the *list of required data items*) is to be collected and the points where it is to be used (i.e., analyzed, reported and/or fed back into the process); also, identify the frequency of collection and the frequency of usage, if other than once at each point.

**Identify Required Data Items (A12)** – Based on the *selected metric definitions and intended usage*, determine what individual pieces of measurement data must be collected to enable calculation of the selected software metrics and use of those metrics as intended.

**Identify Tool Requirements (A212)** – Based on *tool-independent designs of mechanisms* for measurement, identify requirements for automated tools that would implement those mechanisms; this includes not only requirements for measurement tools, but requirements for SEE capabilities that must be provided to support these tools as well.

**Implement Measurement Mechanisms (A3)** – Implement the approved *designs of automated measurement mechanisms* and/or *designs of manual measurement mechanisms*; the implementation of automated measurement mechanisms takes the form of an *instrumented SEE and user documentation*; the implementation of the manual measurement mechanisms takes the form of *published forms and manual procedures.*

**Instrument Software Process (A0)** – Design, implement and execute a combination of automated and manual mechanisms to achieve software measurement objectives; do so within schedule and budget constraints, utilizing available software engineering staff.

**Integrate Measurement Tools (A32)** – Integrate the *acquired measurement tools* with the uninstrumented SEE (and with each other), according to the approved *designs of automated measurement mechanisms*; also, incorporate their user documentation with user documentation for the uninstrumented SEE, to produce user documentation for the resulting instrumented SEE.

**Publish Data Collection Forms (A33)** – Reproduce the approved data collection forms in quantity (so that they will be available for use in collecting measurement data).

**Publish Manual Procedures (A34)** – Published handbooks containing instructions on how to manually collect, analyze, report and/or interpret and feed back measurement data (i.e., *manual procedures* for measurement).

**Purchase Measurement Tools (A311)** – Purchase measurement tools, as specified by the *list of selected measurement tools*, to implement automated measurement mechanisms; some of these tools may be effectively purchased at no cost, if they are public domain or if they are owned by the company for which they are being acquired.

**Select Measurement Tools (A213)** – Evaluate existing *measurement tools* and determine which ones to use to satisfy the *measurement tool requirements*; if not all of the measurement tool requirements can be satisfied using existing tools, select tools that (together) will satisfy as many of these requirements as possible.

**Select Metrics (A11)**– Determine which *candidate software metric definitions* will be used to achieve the *measurement objectives* and the manner in which they will be used to achieve those objectives.

**ICOMs:**

**Acquired Measurement Tools** – Tools (and their associated user documentation) that have been purchased and/or developed specifically to support collection, analysis, reporting and/or feedback of measurement data.

**Analysis Results** – Results of analyzing *collected measurement data.*

**Automation Not Possible** – An output of the *Design Automated Mechanisms* activity that is used as a control (specifically, a trigger) for the *Design Manual Mechanisms* activity; it indicates that some or all of the *data collection and usage requirements* could not be satisfied using automated mechanisms – at least, not within cost constraints.

**Budget** – Budget constraints placed on the overall process of instrumenting a software process (i.e., the *Instrument Software Process* activity).

**Candidate Software Metric Definitions** – Existing definitions of software metrics that may be used to achieve stated *measurement objectives*; it may be that there are alternative definitions of some software metrics; in such cases, customers or corporate organizations may dictate which ones are acceptable candidates, based on their own biases.

**Collected Measurement Data** – All of the information obtained from measurements of software products and processes (including contextual information, such as the date and time that the measurement was performed, the name of the user performing the measurement, etc.).

**Cost Acceptable** – An output of the *Evaluate Cost* activity that is used as a control (specifically, a trigger) for the *Design Automated Mechanisms* activity and/or the *Design Manual Mechanisms* activity; it indicates that the *estimated cost of implementing and sustaining the mechanisms* (as they are defined in the *proposed designs of automated mechanisms* and *proposed designs of manual mechanisms*) is unacceptable, which in turn indicates that the designs of the mechanisms require rework.

**Cost Unacceptable** – An output of the *Evaluate Cost* activity that is used as a control (specifically, a trigger) for the *Approve Design* activity; it indicates that the *estimated cost of implementing and sustaining the mechanisms* (as they are defined in the *proposed designs of automated mechanisms* and *proposed designs of manual mechanisms*) is acceptable.

**Data Collection and Usage Requirements** – Specifications of the individual pieces of information to be collected, the point(s) in the software engineering process where they are to be collected (and the frequency of collection), the manner in which they are to be used, and the point(s) in the software engineering process where they are to be used (and the frequency of use).

**Data Collection Forms** – Paper forms containing blanks where measurement data are to be filled-in manually; a component of both *proposed designs of manual mechanisms* and *designs of manual measurement mechanisms.*

**Designs of Automated Measurement Mechanisms** – The designs of automated mechanisms for collection, analysis, reporting and feedback of measurement data which have been approved for implementation; these designs consist of five components: *tool-independent designs of mechanisms*, *SEE requirements*, *list of selected measurement tools*, *designs of tools to be developed* and *designs of tool integration "glue".*

**Designs of Manual Measurement Mechanisms** – The designs of manual mechanisms for collection, analysis, reporting and feedback of measurement data which have been approved for implementation; these designs consist of two components: *data collection forms* and *manual procedures.*

**Designs of Tool Integration "Glue"** – Designs of software that needs to be developed to integrate measurement tools with each other and with the uninstrumented SEE; a

component of both *proposed designs of automated mechanisms* and *designs of automated measurement mechanisms.*

**Designs of Tools To Be Developed** – Designs of measurement tools that need to be developed to satisfy *measurement tool requirements* that cannot be satisfied by existing tools; a component of both *proposed designs of automated mechanisms* and *designs of automated measurement mechanisms.*

**Estimated Cost of Implementing and Sustaining Mechanisms** – The estimated cost of implementing proposed mechanisms (i.e., *proposed designs of automated mechanisms* and/or *proposed designs of manual mechanisms*) and sustaining operation of those mechanisms throughout the course of the project.

**Feedback** – Information, derived from measurement data, which is to control the process that is being measured (i.e., the software engineering process); feedback may be either manual (e.g., corrective action on the part of a software engineering manager) or automatic (e.g., execution of some automated software engineering tools).

**Instrumented SEE and User Documentation** – The software engineering environment (SEE) after it has been instrumented with automated measurement mechanisms, and accompanying documentation explaining both how to use the SEE to support software engineering activities (including management) and how to use the automated measurement mechanisms.

**Labor Rates** – Hourly rates (i.e., cost) for software engineering labor (specifically, the labor required to implement and sustain the *proposed designs of automated mechanisms* and the *proposed designs of manual mechanisms*).

**List of Required Data Items** – A list of individual pieces of measurement data that need to be collected to calculate and use software metrics in accordance with the *selected metric definitions and intended usage.*

**List of Selected Measurement Tools** – A list of existing measurement tools that have been selected for acquisition (i.e., purchase); a component of both *proposed designs of automated mechanisms* and *designs of automated measurement mechanisms*; in actuality, the selected tools need not be strictly measurement tools, since certain general-purpose tools can sometimes be adapted to perform measurement functions (e.g., a spreadsheet with a reporting capability can be adapted to generate metric reports).

**Manual Procedures** – Instructions on how to manually collect, analyze, report and/or interpret and feed back measurement data; a component of both *proposed designs of manual mechanisms* and *designs of manual measurement mechanisms.*

**Measurement Objectives** – Specific statements of what objectives are to be achieved by software measurement (e.g., control of software quality, improvements in configuration management processes, predictability of software development costs and schedules); these come from numerous sources, including customer requirements and/or expectations, corporate goals and project goals or needs; based on these statements, one can decide what kinds of software metrics to use and how to use them.

**Measurement Tool Requirements** – A set of requirements for *measurement tools.*

**Measurement Tools** – Automated tools used to implement software measurement (i.e., collection, analysis, reporting and/or feedback of measurement data); these need not be tools that are specifically designed for measurement, since certain general-purpose tools can sometimes be adapted to perform measurement functions as well (e.g., a spreadsheet with a reporting capability could be adapted to generate metric reports).

**Metric Reports** – Human readable reports of measurement data and/or analyses of those data; these reports need not be in a textual format (e.g., they may be graphical or tabular).

**Proposed Designs of Automated Mechanisms** – The designs of automated mechanisms for collection, analysis, reporting and feedback of measurement data which have been proposed for implementation; these designs consist of five components: *tool-independent designs of mechanisms*, *SEE requirements*, *list of selected measurement tools*, *designs of tools to be developed* and *designs of tool integration "glue".*

**Proposed Designs of Manual Mechanisms** – The designs of manual mechanisms for collection, analysis, reporting and feedback of measurement data which have been proposed for implementation; these designs consist of two components: *data collection forms* and *manual procedures.*

**Published Forms and Manual Procedures** – *Data collection forms* that have been reproduced in quantity (so that they are available for use in collecting measurement data) and published handbooks containing *manual procedures*; an aggregate that represents implementation of manual measurement mechanisms.

**Schedule** – Schedule constraints placed on the overall process of instrumenting a software process (i.e., the *Instrument Software Process* activity).

**SEE Requirements** – Capabilities that the SEE must provide to support automation of software measurement; a component of both *proposed designs of automated mechanisms* and *designs of automated measurement mechanisms.*

**Selected Metric Definitions and Intended Usage** – Definitions of software metrics that have been chosen for collection, and descriptions of how the metrics are to be used (i.e., analyzed, reported and fed back into the process being measured); the definitions of these metrics are chosen from *candidate software metric definitions.*

**Software Engineering Process Definition** – Descriptions of the software engineering life-cycle processes to which measurement will be applied; although these need not be formal descriptions, the more accurate and detailed they are the easier it is to determine when, where and how to apply measurement.

**Software Engineering Staff** – Personnel engaged in software engineering activities (including software designers, coders, testers, software engineering managers, software process engineers, etc).

**Software Project Data** – All of the information generated by a software project (including software code, documentation, test results, problem reports, project organization charts, schedules, budgets, measurement data, etc).

**Tool-Independent Designs of Mechanisms** – Designs of mechanisms for collection, analysis, reporting and feedback of measurement data which are stated in such a way that they do not depend on the specific tools (if any) used to implement the mechanisms; a component of both *proposed designs of automated mechanisms* and *designs of automated measurement mechanisms.*

**Tool Prices** – Prices of measurement tools that are mentioned in the *proposed designs of automated mechanisms* (specifically, tools that appear in the *list of selected measurement tools*).

**Uninstrumented SEE and User Documentation** – The software engineering environment (SEE) that is to be instrumented, and accompanying documentation explaining how to use it to support software engineering activities (including management); in actuality, since Instrument Software Process could be performed iteratively, this may be a partially instrumented SEE, rather than an uninstrumented SEE (in which case, the result of Instrument Software Process is a more instrumented SEE, rather than just an instrumented SEE).

**Unsatisfied Measurement Tool Requirements** – Those *measurement tool requirements* that cannot be satisfied using existing tools.

### B.3 Amadeus Terminology

The following terms are used to discuss Amadeus.

**Agent** – An executable object (e.g., a program or shell script) that is invoked by the *Amadeus Interpreter* in response to an *event.*

**Amadeus Interpreter** – The component of the Amadeus measurement system that interprets *specifications.* It monitors *events* occurring in the *environment* and invokes *agents* in response, as directed by *specifications.*

**Amadeus Specifier** – The component of the Amadeus measurement system that interprets *specifications*

**Artifact** – An entity (e.g., a software engineering activity, or an object resulting from a software engineering activity) that is subjected to measurement.

**Classification Analysis** – An analysis technique that uses predictive models to predict properties of software.

**Data item** – A piece of information that is required to calculate a *metric.*

**Environment** – Abbreviation for software engineering environment; the totality of computer hardware and software facilities that directly support software engineering.

**Event** – A noteworthy happening that occurs at a particular point in time (e.g., delivery of a source file to a configuration management library).

**Measurement data** – Raw data collected in Amadeus' database; the data is organized as a sequence of data records; each record represents an occurrence of an event or a particular measurement of a metric value; each record consists of a predefined collection of data fields.

**Metric** – A measurable characteristic of an entity (e.g., number of lines of code in a source file, duration of a compiler execution).

**Query** – A request to extract measurement data from Amadeus' database.

**Specification** – An instruction to the *Amadeus Interpreter* concerning the conditions under which a particular *agent* is to be invoked and governing the manner in which the *agent* is invoked; the instruction only takes effect upon activation of the specification (e.g., via an Amadeus_activate command) and ceases its effect upon deactivation of the specification (e.g., via an Amadeus_deactivate command).

**Target class** – The property of software that is to be predicted through *Classification Analysis* (e.g., error proneness).

## C PIP Decomposition

Figure 12 is a depiction of the activity hierarchy into which the PIP is decomposed, represented as a tree. The IDEF0 model of the PIP contains a decomposition page for each non-leaf activity in this tree. These decomposition pages are presented and discussed in Section 2.

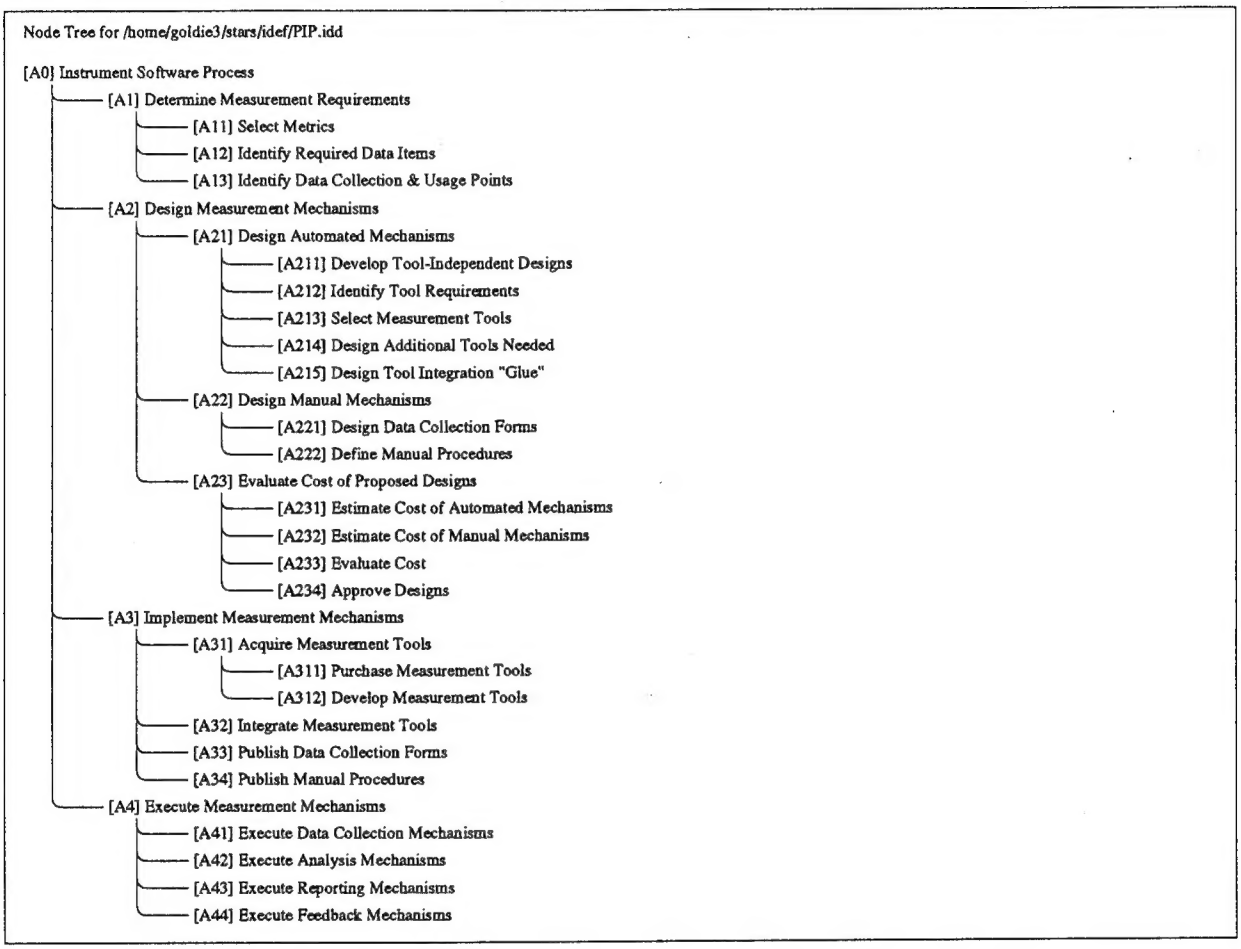



Figure 12: PIP Decomposition Hierarchy